Final Report

Department of Computer Science

Calvin University

LoTide: An Adaptive Music Workflow

Mark Wissink, Matthew Nykamp, Ian Park

Date: 4/26/20

Mentor: Joel Adams

Honors Project: No

Project Vision and Overview

Video games products and similar media have been a driving force behind many advances in technology. Graphics cards, virtual reality, and powerful software are continually becoming more powerful to accommodate a better experience of gaming. However, the influence of video games has also extended to other media such as art and music. Adaptive music is a recent paradigm of composition that accounts for the non-linearity of video games. Using techniques like horizontal scaling, a song can slowly transition into another song; and vertical techniques can reorchestrate the current song. These techniques make for a more immersive experience as the soundtrack adapts to a player's surroundings.

Creating adaptive music is a non-trivial problem, and existing products provide software solutions to compose this type of music for games. However, most of these existing solutions emphasize horizontal techniques, shifting between pre-rendered layers at runtime. There seems to be a small disconnect between the digital audio workspace (DAW) and the runtime API for adapting the music. Many vertical techniques suffer from this disconnect because pre-rendered audio does not provide that flexibility. The vision behind Lotide is to provide even more flexibility with audio in nonlinear use cases by providing an interface to a DAW that plays music in real-time. By providing a cross-platform audio backend, developers will be able to interface through a language agnostic API to manipulate and play music in their program. Another important aspect to our project was the licensing that is associated with our software. Most existing

solutions have proprietary licensing and are not open-source. Our vision is a FOSS (Free and open-source software) solution to adaptive music.

Background

Nonlinear media usually takes form in the shape of video games. By nonlinear media, we refer to the type of media which is not necessarily static, as a movie soundtrack would be, but rather an auditory experience which can be different depending on external variables. This is commonly called adaptive music. For example, in the final lap of Mario Kart, the music is pitch-shifted to a higher than normal value and the tempo is increased, resulting in a more suspenseful overall feel. In this way, the variable lastLap (as it might be implemented in Mario Kart) determines what type of music is played.

Another example is sound effects. Many games have sound effects which are played when a specific action or event occurs. Attacking with your character might play an accompanying sound of a sword whirring through the air, or moving the character might produce a footstep sound. In this way, the overall auditory experience is changed when some external factors change.

One downside to these existing methods of implementing adaptive music is that blending different segments of rendered audio can be jarring - this can be seen in the Mario Kart example. When you cross the finish line, the sound immediately jumps to a small transitional sound effect, and then immediately jumps back to the pitch shifted sound, probably to avoid some form of auditory artifact caused by either an immediate jump to the pitch shifting or a slower ramping up of the pitch and tempo.

Another downside of existing solutions is that they scale poorly. If a developer wishes to have a soundtrack that reacts to the game—by introducing a new instrument on top of the existing sound, that would be easily manageable by creating a set of rendered audio for each instrument, and then blending the new audio in. However, having a different segment of rendered audio for each instrument in a song quickly becomes unmanageable as the number of instruments increases, not only in terms of performance (playing 25 or more rendered audio at the same time could cause problems), but also in terms of managing which should be playing or not be playing at any given time.

We decided early on that many sound effects in games should be immediate, and probably don't need to be involved in the music of the game. In both examples earlier, having the sound effect play immediately is important. However, our system would support a sound effect being played in time with the music, if appropriate. But, we didn't design with this in mind - our focus is mostly on providing a way to create music.

The essential problem seemed to be that the parts that made up the music of many games was inaccessible to programmers of the game - all they had access to was rendered audio. They overcame the limitations of rendered audio in a number of ways, as we shall see shortly, but we decided that creating a system which allows a programmer control over these elements would allow for even more possibilities. Having a way to easily add or remove certain musical layers, either horizontally (changing what notes the instruments might be playing) or vertically (changing which instruments are

playing entirely, or switching a set of notes from one instrument to another) seemed like the way to go.

So, we decided that we should begin looking at tools that game developers have been using to satisfy these problems in order to figure out what exactly our proposed workflow would look like, and if it was really needed. From our research, we found FMOD and Elias to be the popular options for adaptive music. With plugin support on industry standards like Unity and Unreal, these programs had a lot of offer. The innovation that LoTide makes on these products is real-time audio rendering. With this method, users are able to manipulate composition data at runtime. Unlike FMOD and Elias which primarily use pre-rendered audio, LoTide allows users to rearrange notes to change tempo, match key, and generate notes on the fly. Another factor to consider is that both softwares, FMOD and Elias, are proprietary, closed source software. When looking for FOSS solutions, we came across OAML (Open Adaptive Music Library). Similar to FMOD and Elias, the solution to dynamic in OAML is to compose music out of pre-rendered loops. While this solution does work well, our goal with LoTide is to take adaptive music a step further with dynamic sound and composition at runtime.

Implementation and Design

TSAL (Thead Safe Audio Library)

Why TSAL?

TSAL is a project created by Joel Adams and Mark Wissink to complement the existing TSGL (Thead Safe Graphics Library); both are tools that can be used in the instruction of the parallel computing paradigm. TSAL provides a minimal implementation

of a DAW that can be used to generate audio given notes to play. TSAL was a natural choice as a backend for LoTide since it both inspired and fit the specifications for the project. Most importantly, TSAL provides a simple, lightweight DAW that can do real-time audio generation which supports the goals of the adaptive music framework. Also, using TSAL in a full-fledge project was a good test of the functionality that TSAL provided, and it gave insights into what needed to be improved.

Audio Backend

TSAL had originally used RtAudio as it's solution to cross-platform audio library. While RtAudio worked well, it was primarily chosen as it was easy to incorporate into the build system early on. However, RtAudio is not actively developed and lacks good documentation. In its place, PortAudio was chosen since it is actively developed and used as a solution in many other projects. The audio library has always been tightly coupled to the TSAL Mixer class, and switching between the libraries was trivial in the code. However, successfully adding PortAudio to the build system was a little more complex.

Another issue in the original development of TSAL was the lack of support for stereo (2+ channel) audio. Initially developing with mono (1 channel) audio made the project more maintainable, but it was an oversight in future uses of the library. With switching the audio backend, it was a natural time to add stereo support to the library. To handle the logic of stereo audio, the AudioBuffer class was created. AudioBuffer handles the logic of multiple channels and frames. It also handles the logic behind

interleaving. Interleaved audio stores audio samples in a single buffer grouped by frame. In a stereo example, the first two values in the buffer are the left and right channel samples for the first frame, the next two values being the left and right channel samples for the second frame, and so on. Rather than having multiple buffers in memory representing each channel, interleaved audio takes advantage of the spatial locality in the CPU cache since audio frames are generally accessed sequentially.

Thread Safety

The thread safety functionality is a core part of TSAL since it's goal is to provide a library that can easily handle complexities of multi-threading. The initial approach to solve this problem was for each device to provide thread-safe operations for adding new devices. If the model of TSAL can be thought of as a graph with the Mixer at the root, each node has to provide thread-safe operations for adding new nodes. The main problem with this implementation is that it can cause a buffer underrun in the underlying audio output. For example, if the audio library requests new samples from the Mixer class, it could fail to receive samples in time in the case that some Channel on the Mixer is locked because of adding new devices. The solution implemented was to have a single lock on the model that would ensure changes to the model where made after audio had been generated for the current audio request. Now Channels and other devices that route audio can make asynchronous requests to the Mixer to make changes to the model.

Plugins

One the goals for LoTide and TSAL was to have plugin support for virtual instruments and effects. This would give developers access to a wide array of different sounds rather than just the native TSAL instruments. Some time was dedicated to researching what type of plugins could be supported—most being some form of dynamic libraries. One consideration we had to make in the case of LoTide was the licensing on the software plugin. Since LoTide generates audio at runtime, whatever plugins used to generate the audio would have to be bundled with the project file. And since these files are expected to be distributed to many users, this could potentially be a violation of many redistribution clauses on proprietary plugins. Once again, we turned to the FOSS solution of LADSPA (Linux Audio Developer's Simple Plugin API) and LV2 (LADSPA Version 2). However, loading dynamic libraries in a cross-platform environment proved to be a difficult task. A successful implementation was made for Linux but not for Windows.

Synthesizer Improvements

As an alternative to the incompleted plugin support, we focused on extending the functionality of the PolySynth synthesizer in TSAL. At the start of the project, PolySynth only supported sine, saw, and square waveform outputs with an amplitude envelope. To make PolySynth robust, many new features where added including white noise, filters, phase modulation, frequency modulation, and LFO (low frequency oscillator). White noise was an addition to the Oscillator class which generates random values as output.

The Filter class implements an algorithm that works as a low-, high-, and band-pass filter. The phase and frequency modulation where more additions to the Oscillator that manipulate the output waveform by some given input, usually another waveform. The LFO is simply an oscillator that can be used to slowy modify some value such as a volume or frequency cutoff over time, making the sound more alive.

Device Parameters

While developing the frontend of LoTide, it became clear that the way parameters where being handled on devices could be improved. Before, each parameter had its own setter and getter. While this works fine in code, it can be cumbersome when designing interfaces for these systems. An index based parameter system, similar to the LADSPA plugins, seemed to be a good solution. ParameterManager is the implementation of that system, providing the necessary methods to define and access parameters on devices. By using enums as an alias for the index, programmers can easily read what parameters they are modifying and safely make changes in code.

Lotide Backend

While we had TSAL as a solid base for generating the audio, we realized quickly that we would need many structures forming a layer of abstraction in order to create a usable library. While TSAL has a PolySynth, for example, we needed to provide a way for a user to specify what a song should look like without manually managing the calls to play and stop for each note.

In essence, we wanted the user to be able to specify a song, which we ended up thinking of as a collection of instruments playing some collection of notes. We wanted the user to be able to not only start or stop a piece of music, but interact with it in more ways. To facilitate this, notes were stored in collections known as phrases. Under the new schema, the song would be composed of a number of phrases, where each instrument plays some phrase at a given time.

As our music is adaptive, we required additional information regarding which phrase each instrument would play given the current time. To do this, we created a grouping, which simply maps each instrument to a list of phrases to play. By creating new groupings and telling the song to switch to a new group, the user can now create almost any type of non-linear music at the behest of outside inputs.

The last piece of the puzzle comes in the form of a sequencer, which periodically asks the song which notes should be playing. It plays those notes by telling the TSAL instrument to play it. It remembers which notes are playing, and when they started, so it can at the appropriate time tell the synthesizer to stop playing the notes. When the main lotide object is told to start playing, it creates a new thread to facilitate this, allowing for other lotide interactions to occur simultaneously to the playing of a song.

With all of these structures in place, it is now easy for a user to simply swap out what is being played by certain instruments at runtime - a single function call in the game code will do the trick. In addition, to define new composition at runtime is possible, and some other small benefits present themselves as well. For instance, the

ability to scale our music's pitch up or down is trivial, and presents no artifacts as rendered audio would.

One potential issue with our system is that the complexity of instruments and effects is limited to what can be processed in real time. Considering that some professionally created music commonly takes render times on the order of minutes, these types of sounds may be impossible in our real time context, and would also be user-dependent (so a song that would work well on a really nice computer might work poorly on a low-spec one). Discovering the bounds of our system is definitely an important area to explore should Lotide be proven useful. Given that our target audience is primarily indie game developers who lack the budget for other commercial grade sound software, this may be an acceptable downside.

Cross-Platform

The Build System

In order to obtain cross platform support, we had to use a build system for simplifying the build process and detecting the platform. The main branch of TSAL was using a workflow that incorporated the traditional GNU build tools. This worked well in UNIX environments but required MinGW to work on Windows. These tools were not as reliable and straightforward as the official build tools offered by Microsoft through Visual Studio. We also had to decide between using a compatibility layer on top of the operating system or writing portable C++ code and compile natively on our target platforms. While using a compatibility layer would have made the code less complex, it

would come at a performance cost. Since LoTide and TSAL would be running in a real-time system where execution time is important, we decided against a compatibility layer in favor of a native compilation approach. So, we ported the project to CMake, which offered build outputs targeting the Windows compiler and GCC on unix systems.

We tried to utilize many modern CMake features such as embedding external library dependencies as git submodules and automatically generating packages for Linux distributions, namely Debian(deb) and RedHat(rpm). Furthermore, we made it so that our library would also be easily consumed by other modern CMake projects just by invoking an `add_subdirectory(<directory>)` to our project directory.

Although the convenience was beneficial once we got it set up, we faced a lot of small problems with getting CMake to work consistently. The downside was that the official documentation for CMake was not so clear. Likewise, examples for cross-platform implementations were lacking.

Issues with OS Versioning

One of the challenges with making the software cross platform was that some bugs were not easily reproducible even among the same Windows systems. Small changes to the version of Windows itself produced different build results. Likewise, bugs on the sound output were difficult to reproduce and debug.

Serialization and File Format

Initially, we thought that our system would support Midi files and Soundfonts to incorporate them into our platform. So, we tried to find out ways to generate a compressed file containing all of these files. However, for simplicity, we omitted support for those files and chose to serialize the state of a song and provide that as a file format. At first we attempted to use Boost for our serialization because it had a pretty straightforward way of serializing native types. However, because the Windows build was cumbersome, we switched to a more modern serialization library called cereal. Since this project was compatible with CMake, we were able to easily embed this within our project.

API

POSIX Sockets/WinSock

Along with our C++ library, we wanted another way for programmers to consume our library. This was primarily due to the fact that game development often happens in other languages such as C# and Java. However, an actual port of Lotide to all of those platforms seemed like an inordinate amount of work. So, we decided to create a standardized API through a Remote Procedure Call (RPC) server. By doing so, we could send remote procedures (i.e. function names with parameters) from a client and send back a state of the system without providing any other language abstractions.

At first, we researched some pre-existing RPC projects that we might use. However, many of them had their own protocols in place, which felt like a potential overhead for us and for the users of our API. So, we decided to start from scratch and

build an RPC using TCP sockets and JSON with our own protocol. So we made a client that sent a JSON string with a UUID and the procedure specifications (command and parameters) to the server. Then, the server also sent the serialized state of the LoTide instance to the client of the specified UUID. We also tried to multi-thread the server so that it could future proof it. There were two cases of the multi threaded server. The first was during development, a developer could want a running instance of the game while creating the music in a different instance or while monitoring the instance for debugging. In this case, each client would have to share the state of the LoTide instance with another client. So a UUID would identify a client and associate it with an instance of LoTide running from a different thread. And each procedure call is run by a thread generated on initialization in a thread pool.

Clients

Previous Efforts

We began our process of building a GUI editor which was planned to look much like existing editors. While investigating which framework would be ideal for this project, we initially began with QT. Our reasoning at the time was that it was widely accepted and used in many commercial and open source projects, including LMMS. At the time, it seemed to be worth the effort to learn a brand new framework in order to create a professional looking UI.

However, as we began the exploratory process of building the UI for our project, we realized that the process of learning QT was more like learning an entirely new

language rather than just using a GUI framework - we realized that if we wanted our main deliverable to be as developed as we would like, a simpler solution would be preferable.

Why Web?

We eventually decided to build our proof-of-concept prototype using web technologies because it was easy to implement our ideas quickly without much boilerplate. Furthermore, since we are targeting cross platform support, we do not have to worry about native toolkits with the web.

WebSockets and Node.js

Since we decided to build web apps for our test/prototype clients, we had to find a way to use our API through the web. The only way that our daemon could actively respond to our web app was through the Web Sockets standard. Although we could host Web Sockets directly from C++, we decided, as a proof-of-concept, to use Node.js as the primary Web Sockets server to show that we could run our app using various languages that support sockets. Node.js had socket support built into the language. So, it was very convenient to make it work with our socket implementation. In addition to our prototype, we have CLI client implementations to test each layer in our protocol.

Results and Discussion

TSAL

Many improvements were made to TSAL over the course of the project. When it was initially created during the summer, some shortcuts were taken to simplify the

project. Changes in the audio backend and multi-channel support have made TSAL a viable option for synthesizing audio. New structures and bug fixes in the code base made TSAL much more stable while simplifying the API. Adding LADSPA and LV2 plugins proved to be a complicated task, but it may be worth the effort if support were added in the future. Some options to make the task feasible would be to use a shared library loader such as Boost.DLL or QLibrary. Writing a cross-platform shared library loader from scratch is a possible solution, but it may be smarter to use an existing solution. Many new additions to PolySynth made the class a functional synthesizer capable of generating a wide array of noise. With additions of filter, modulation, and LFO, the virtual instrument proves to be a viable option for creating music. Finally, the ParameterManager class created an effective method for access and connecting device parameters to both interfaces and users.

LoTide

We were able to successfully achieve our goals in implementing a system for dynamic music generation and consumption. The sequencer correctly manages the logic among groups, phrases, and synthesizers. Also, we have been able to achieve cross-platform support. Likewise, we were able to support multiple languages through a socket RPC. Furthermore, we have been able to demonstrate a couple of use cases through our frontend designs.

Conclusion

This project proved to be a challenge with the robust requirements we set for ourselves. Achieving a cross-platform, multi-language framework proved to be difficult

throughout the entire lifetime of the project. However, we were able to build a complete version of LoTide that met our basic expectations. Starting with TSAL, we were able to build a more stable environment that functioned closely to a modern DAW. There were many quality of life improvements and new features that made their way into TSAL. The LoTide system successfully integrated our model of a sequencer, notes, phrases, groups, and songs effectively to produce the dynamic music structure that we envisioned. With the implementation of serialization, LoTide can produce portable .lot files to be used in projects. Near the end of the project cycle, we finished off with an example implementation of LoTide in a simple game, and we also created a demo client that can be used to interface with our system. Overall, our project was quite successful, but there are many missing features and improvements that could make the system more viable. We hope the system continues to mature and grow in the future.

Future Work

The future goals of both TSAL and LoTide are closely aligned as we worked towards a fully featured DAW. Over the course of our project, we have implemented an MVP (minimum viable product) with lots of room for growth.

TSAL

The main addition to TSAL that would greatly benefit it and LoTide alike would be the support for virtual instrument and effect plugins. Expanding the collection of native effects and instruments would also be important since it gives new users more options if they don’t have access to many audio plugins. Another reason to add more native effects and instruments is to simply have more examples of implementations in the code

base. With new students or developers working in the code, this would become an important aspect.

One weakness of TSAL is that the Mixer class is tightly coupled to the audio backend; it would be good to separate the Mixer and the audio backend. Similar to LMMS (Linux MultiMedia Studio), multiple audio backends can be used and selected by the user depending on what their machine supports. With this flexible method, if a better audio backend is discovered, it can simply be added as a new backend alongside any existing backends. In the context of LoTide, this feature would also be important since it could be running on many different platforms that may need multiple different audio backends to be supported.

LoTide

LoTide is a flexible system which allows for a variety of improvements. One area we want to pursue is algorithmically generated music. By using some music theory, we could algorithmically generate a smooth transition between two different songs. Whether this is done through an AI or smart algorithm, generating music would be very useful in the case of transitions or melodys. Either statically or dynamically generating would take a load off the composer in terms of how much music they would have to create.

Much of the interface created for LoTide in this project was proof-of-concept. While we did build a sophisticated web interface, future work would include a native, cross-platform client. By using a toolkit like Qt, we would be able to fast, native GUIs on all platforms. One feature that was missing from our demo client was a comprehensive

project view. Since our songs are broken down into groups and phrases, a view that supports such an interface is important for cleanly navigating the project structure.

Acknowledgements

On behalf of the whole team, we would like to thank Professor Adams, the CS department, and friends and family. Professor Adams was the main advisor for our project, and he helped us throughout the whole process. He provided useful insight into our problems and questions and kept us motivated while working from home. The CS department was also supportive in their flexibility of expectations and deadlines given the circumstances. Our friends and family supported us along the way, and this project wouldn't have been possible without them.

References

LoTide

FMOD https://www.fmod.com/

OAML https://github.com/oamldev/oaml

Elias https://www.eliassoftware.com/

TSAL

LMMS https://lmms.io/

Ardour https://ardour.org/

MusicDSP https://www.musicdsp.org/en/latest/

Appendixes

```
#include <iostream>

#include "LoTide.hpp"
#include "LTSynth.hpp"
#include "Group.hpp"
#include "Phrase.hpp"
#include "Sequencer.hpp"
#include "Note.hpp"

#include "Server.hpp"

using namespace lotide;

int main()
{
	Server serv(8198);
	serv.init();
}
add_executable(testClient testClient.cpp)
target_link_libraries(testClient LoTide)

add_executable(testServer testServer.cpp)
target_link_libraries(testServer LoTide)
#include <iostream>
#include <regex>

#ifdef _WIN32
	#include <winsock2.h>
	#include <Ws2tcpip.h>
#else
	#include <sys/socket.h>
	#include <arpa/inet.h>
	#include <stdlib.h>
	#include <unistd.h>
	#include <cstring>
#endif

int parseCommand(std::string);
int socketStart(std::string);
```

```
int main() {
	std::cout << "Started LoTide CLI.\n" \
			<< "Enter LoTide Commands.\n" \
			<< "Type \"help\" for a list of commands.\n" \
			<< std::endl;

	for(;;) {
		std::string userInput;
		std::cout << "lotide> " << std::flush;
		std::getline(std::cin, userInput);

		if(parseCommand(userInput) == 1) {
			break;
		}
	}

	return 0;
}

int socketStart(std::string response) {
#ifdef _WIN32
	WSADATA wsa_data;
	WSAStartup(MAKEWORD(1,1), &wsa_data);
#endif


	unsigned int sock = 0, valread;
	struct sockaddr_in serv_addr;
	char buffer[4096] = {0};

	// TODO Consider AF_UNIX because it is just a file
	sock = socket(AF_INET, SOCK_STREAM, 0);
	if (sock == -1) {
		perror("cannot create socket");
		return -1;
	}

	// NOTE Clear address (basically like null assignment)
	memset(&serv_addr, 0, sizeof serv_addr);

	serv_addr.sin_family = AF_INET;
	serv_addr.sin_port = htons(8198);
```

```
	inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr);
	if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) == -1) {
		perror("connect failed");
		close(sock);
		return -1;
	}

	// TODO Have to handle string length
	send(sock , response.c_str(), 8192, 0);
	// send(sock , response.c_str(), response.length(), 0);
	std::cout << "Message sent" << std::endl;
	std::cout << "From socketStart() this is c_str" << response.c_str() << std::endl;

	valread = recv(sock , buffer, sizeof(buffer), 0);
	std::cout << buffer << std::endl;

#ifdef _WIN32
	closesocket(sock);
	WSACleanup();
#else
	shutdown(sock, SHUT_RDWR);
	close(sock);
#endif

	return 0;
}

int parseCommand(std::string s) {
	std::istringstream commandStream(s);
	int count = 0;
	std::vector<std::string> commands;

	do {
		std::string command;
		commandStream >> command;

		commands.push_back(command);

		++count;
	} while (commandStream);

	std::string command1 = commands[0];
```

```
        if(count >= 3) {
                std::regex quote("\"");
                std::string command2 = std::regex_replace(commands[1], quote, "");
        }

        if (command1 == "exit") {
                std::cout << "Exitting..." << std::endl;
                return 1;
        } else {
                if (command1 == "help") {
                        std::cout << "load \"[FILEPATH]\"\t Load .lot file by path.\n" \
                                        << "play \"[TRACKNAME]\"\t Play listed track.\n"     \
                                        << std::endl;
                } else if (command1 == "load") {
                        if (count != 3) {
                                std::cout << "Enter file name." << std::endl;
                        } else {
                                std::cout << "Loading file.." \
                                                << std::endl;
                        }
                } else if (command1 == "play") {
                        socketStart("{ \"command\": \"play\", \"parameters\": [\"\"] }");
                } else if (command1 == "playNote") {
                        socketStart("{ \"command\": \"playNote\", \"parameters\": [\"\"] }");
                } else if (command1 == "close") {
                        socketStart("{ \"command\": \"close\", \"parameters\": [\"\"] }");
                        exit(EXIT_SUCCESS);
                } else if (command1 == "stop") {
                        socketStart("{ \"command\": \"stop\", \"parameters\": [\"\"] }");
                } else if (command1 == "addSong") {
                        std::string begin = "{ \"command\": \"addSong\", \"parameters\": [\"";
                        std::string end = "\"] }";
                        std::string moreInput;
                        std::cout << "Song Name> " << std::flush;
                        std::getline(std::cin, moreInput);
                        std::string outString = begin + moreInput + end;
                        socketStart(outString);
                } else if (command1 == "addSynth") {
                        socketStart("{ \"command\": \"addSynth\", \"parameters\": [\"\"] }");
                } else if (command1 == "addGroup") {
                        std::string begin = "{ \"command\": \"addGroup\", \"parameters\":
[\"";
                        std::string end = "\"] }";
```

```
                        std::string moreInput;
                        std::cout << "Group Name> " << std::flush;
                        std::getline(std::cin, moreInput);
                        std::string outString = begin + moreInput + end;
                        socketStart(outString);
                } else if (command1 == "addPhrase") {
                        std::string begin = "{ \"command\": \"addPhrase\", \"parameters\":
[\"";
                        std::string end = "\"] }";
                        std::string moreInput;
                        std::cout << "Phrase Name> " << std::flush;
                        std::getline(std::cin, moreInput);
                        std::string outString = begin + moreInput + end;
                        socketStart(outString);
                } else if (command1 == "addNote") {
                        std::string begin = "{ \"command\": \"addNote\", \"parameters\": [\"";
                        std::string end = "\"] }";

                        std::string moreInput1;
                        std::cout << "Note> " << std::flush;
                        std::getline(std::cin, moreInput1);

                        std::string moreInput2;
                        std::cout << "Velocity> " << std::flush;
                        std::getline(std::cin, moreInput2);

                        std::string moreInput3;
                        std::cout << "StartTime> " << std::flush;
                        std::getline(std::cin, moreInput3);

                        std::string moreInput4;
                        std::cout << "Duration> " << std::flush;
                        std::getline(std::cin, moreInput4);
                        std::string outString = begin + moreInput1+"\"," +
"\""+moreInput2+"\"," + "\""+moreInput3+"\"," + "\""+moreInput4 + end;

                        socketStart(outString);
                } else if (command1 == "removeNote") {
                        std::string begin = "{ \"command\": \"removeNote\", \"parameters\":
[\"";
                        std::string end = "\"] }";
                        std::string moreInput1;
                        std::cout << "Note> " << std::flush;
```

```
                        std::getline(std::cin, moreInput1);

                        std::string moreInput2;
                        std::cout << "Velocity> " << std::flush;
                        std::getline(std::cin, moreInput2);
                        std::string outString = begin + moreInput1+"\"," + "\""+moreInput2 +
end;

                        socketStart(outString);
                } else if (command1 == "setActivePhrase") {
                        std::string begin = "{ \"command\": \"setActivePhrase\",
\"parameters\": [\"";
                        std::string end = "\"] }";
                        std::string moreInput;
                        std::cout << "Active Phrase> " << std::flush;
                        std::getline(std::cin, moreInput);
                        std::string outString = begin + moreInput + end;
                        socketStart(outString);
                } else if (command1 == "setActiveGroup") {
                        std::string begin = "{ \"command\": \"setActiveGroup\",
\"parameters\": [\"";
                        std::string end = "\"] }";
                        std::string moreInput;
                        std::cout << "Active Group> " << std::flush;
                        std::getline(std::cin, moreInput);
                        std::string outString = begin + moreInput + end;
                        socketStart(outString);
                } else if (command1 == "setLength") {
                        std::string begin = "{ \"command\": \"setLength\", \"parameters\":
[\"";
                        std::string end = "\"] }";
                        std::string moreInput;
                        std::cout << "Length> " << std::flush;
                        std::getline(std::cin, moreInput);
                        std::string outString = begin + moreInput + end;
                        socketStart(outString);
                } else if (command1 == "setActiveSong") {
                        std::string begin = "{ \"command\": \"setActiveSong\",
\"parameters\": [\"";
                        std::string end = "\"] }";
                        std::string moreInput;
                        std::cout << "Active Song> " << std::flush;
                        std::getline(std::cin, moreInput);
```

```
                std::string outString = begin + moreInput + end;
                socketStart(outString);
            } else {
                std::cout << "Command \"" << command1 << "\" is not found!\n" \
                          << "Type \"help\" for a list of commands.\n"    \
                          << std::endl;
            }

            return 0;
        }
}
cmake_minimum_required(VERSION 3.1...3.15)

if(${CMAKE_VERSION} VERSION_LESS 3.12)
    cmake_policy(VERSION
${CMAKE_MAJOR_VERSION}.${CMAKE_MINOR_VERSION})
endif()

set(LOCAL_PROJECT_VERSION    "0.0.1")
set(LOCAL_PROJECT_VENDOR     "Calvin University")
set(LOCAL_PROJECT_NAMESPACE  "LoTide")
set(LOCAL_PROJECT_NAME       "lotide")
set(LOCAL_PROJECT_OUTPUT_NAME "lotide")
set(LOCAL_PROJECT_DESCRIPTION "LoTide DAW")

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)


# Set the project name
project(${LOCAL_PROJECT_NAME}
            VERSION ${LOCAL_PROJECT_VERSION}
            DESCRIPTION ${LOCAL_PROJECT_DESCRIPTION}
            LANGUAGES C CXX)

find_package(Git QUIET)
if(GIT_FOUND AND EXISTS "${PROJECT_SOURCE_DIR}/.git")
# Update submodules as needed
    option(GIT_SUBMODULE "Check submodules during build" ON)
    if(GIT_SUBMODULE)
        message(STATUS "Submodule update")
```

```
                execute_process(COMMAND ${GIT_EXECUTABLE} submodule update --init
--recursive  --remote
                                                WORKING_DIRECTORY
${CMAKE_CURRENT_SOURCE_DIR}
                                                RESULT_VARIABLE GIT_SUBMOD_RESULT)
                if(NOT GIT_SUBMOD_RESULT EQUAL "0")
                        message(FATAL_ERROR "git submodule update --init failed with
${GIT_SUBMOD_RESULT}, please checkout submodules")
                endif()
        endif()
endif()

if(NOT EXISTS "${PROJECT_SOURCE_DIR}/extern/TSAL")
        message(FATAL_ERROR "The submodules were not downloaded!
GIT_SUBMODULE was turned off or failed. Please update submodules and try again.")
endif()

if(NOT EXISTS "${PROJECT_SOURCE_DIR}/extern/cereal")
        message(FATAL_ERROR "The submodules were not downloaded!
GIT_SUBMODULE was turned off or failed. Please update submodules and try again.")
endif()

if(NOT EXISTS "${PROJECT_SOURCE_DIR}/extern/json/")
        message(FATAL_ERROR "The submodules were not downloaded!
GIT_SUBMODULE was turned off or failed. Please update submodules and try again.")
endif()

list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_LIST_DIR}/cmake")


# list(APPEND CMAKE_PREFIX_PATH "/opt/grpc" "/opt/protobuf")

# find_package(Protobuf REQUIRED)
# find_package(GRPC REQUIRED)

add_subdirectory(tests)

add_subdirectory(extern)

add_subdirectory(src)
add_subdirectory(TSAL)

set(SKIP_PORTABILITY_TEST ON CACHE INTERNAL "")
```

```
set(SKIP_PERFORMANCE_COMPARISON ON CACHE INTERNAL "")
set(JUST_INSTALL_CEREAL ON CACHE INTERNAL "")
add_subdirectory(cereal)

set(JSON_BuildTests OFF CACHE INTERNAL "")
add_subdirectory(json)
#include "LTSynth.hpp"

namespace lotide {
	LTSynth::LTSynth() {}
	// LTSynth::LTSynth(unsigned id, tsal::Mixer &mixer) : mSynth(&mixer), mId(id) {
	LTSynth::LTSynth(unsigned id) : mId(id) {
	}
}
#include "Song.hpp"

namespace lotide {
	Song::Song() {}

	Song::Song(std::string name, tsal::Mixer& m, int tempo) : mName(name),
mTempo(tempo) {
		mMixer = &m;
		activeGroup = NULL;
		mCurrentLength = 0;
	}

	Song::Song(Song&& other) noexcept :
		mName(std::move(other.mName)),
		mSynths(std::move(other.mSynths)),
		mPhrases(std::move(other.mPhrases)),
		groups(std::move(other.groups)),
		activeGroup(std::move(other.activeGroup)),
		mCurrentLength(other.mCurrentLength),
		mNextUniqueSynthId(other.mNextUniqueSynthId) {
		mMixer = other.mMixer;
		mTempo = other.getTempo();
	}

	Group& Song::makeNewGroup(std::string groupName) {
		Group g(groupName, getSynthIds());
		groups.push_back(std::move(g));

		if (groups.size() == 1) {
```

```
                setGroup(groups[0]);
            }

            return groups[groups.size() - 1];
        }

        std::unordered_map<unsigned, std::list<Note>> Song::getUpcoming(unsigned
time) {
            std::unordered_map<unsigned, std::list<Note>> notes;

            if (mCurrentLength == 0) {
                return notes;
            }

            unsigned normTime = time % mCurrentLength;

            if (normTime == 0 && nextGroup != NULL) {
                setGroup(*nextGroup);
            }

            for (auto& kv : mSynths) {

                std::vector<unsigned>& phrases =
activeGroup->getPhrases(kv->getId());

                if (phrases.size() != 0) {

                    int desId = 0;
                    Phrase* desired = &mPhrases[phrases[desId]];

                    bool noPhrase = false;

                    unsigned previousLength = 0;
                    while (desired->getLength() + previousLength <= normTime) {
                        desId++;

                        if (desId >= phrases.size() || phrases[desId] >=
mPhrases.size()) {
                            noPhrase = true;
                            break;
                        }

                        previousLength += desired->getLength();
```

```
                        desired = &mPhrases[phrases[desId]];
                    }

                    if (!noPhrase) {

                        unsigned actualTime = normTime - previousLength;

                        for (auto& note : desired->getNotes()) {
                            if (note.getStartTime() == actualTime) {
                                notes[kv->getId()].push_back(note);
                            }
                        }
                    }
                }
            }

            return notes;
}

std::vector<LTSynth*> Song::getSynths() {
            std::vector < LTSynth* > synths;

            for (auto& kv : mSynths) {
                LTSynth* ptr = kv.get();
                synths.push_back(ptr);
            }

            return synths;
}

std::vector<unsigned> Song::getSynthIds() {
            std::vector<unsigned> synths;

            for (auto& kv : mSynths) {
                synths.push_back(kv->getId());
            }

            return synths;
}

LTSynth& Song::addSynth() {
            mSynths.push_back(std::make_unique<LTSynth>(mNextUniqueSynthId));
            LTSynth& newSynth = *mSynths[mNextUniqueSynthId];
```

```
		mNextUniqueSynthId++;

		for (Group g : groups) {
			g.addSynth(newSynth.getId());
		}

		mMixer->add(newSynth.getSynth());
		return newSynth;
	}

	void Song::setGroup(std::string name) {
		activeGroupName = name;
		for (Group& g : groups) {
			if (g.getName() == name) {
				setGroup(g);
			}
		}
	}

	void Song::setGroup(Group& g) {
		activeGroup = &g;

		mCurrentLength = 0;

		for (auto& kv : mSynths) {
			std::vector<unsigned> phrases =
activeGroup->getPhrases(kv->getId());

			unsigned thisLength = 0;

			for (auto& phrase : phrases) {
				thisLength += mPhrases[phrase].getLength();
			}

			if (thisLength > mCurrentLength) {
				mCurrentLength = thisLength;
			}
		}

		if (g.getNextGroup() == NULL) {
			nextGroup = NULL;
		} else {
			nextGroup = &groups[g.getNextGroup()];
```

```
		}
	}

	Phrase& Song::addPhrase(std::string name) {
		Phrase p(name, mNextUniquePhraseId);
		mPhrases.push_back(std::move(p));

		mNextUniquePhraseId++;

		return mPhrases[mPhrases.size() - 1];
	}

	void Song::setNextGroup(std::string name) {
		for (Group& g : groups) {
			if (g.getName() == name) {
				setNextGroup(g);
			}
		}
	}

	void Song::setNextGroup(Group& g) {
		nextGroup = &g;
	}

	Group& Song::getActiveGroup() {
		return *activeGroup;
	}

	Phrase& Song::getPhrase(unsigned phraseId) {
		return mPhrases[phraseId];
	}

	LTSynth& Song::getSynth(unsigned synthId) {
		return *mSynths[synthId];
	}
}
#include "Group.hpp"

namespace lotide {
	Group::Group() {}
	Group::Group(std::string name, std::vector<unsigned> songSynths) :
mName(name) {
		for (unsigned id : songSynths) {
```

```
                mSynthPhrases[id] = std::vector<unsigned>();
            }
        }

        void Group::addPhrase(unsigned synthId, unsigned phraseId) {
            mSynthPhrases[synthId].push_back(phraseId);
        }

        std::vector<unsigned>& Group::getPhrases(unsigned synthId) {
            return mSynthPhrases[synthId];
        }

        void Group::addSynth(unsigned synthId) {
            mSynthPhrases[synthId] = std::vector<unsigned>();
        }

        void Group::addNextGroup(int groupId) {
            nextGroup = groupId;
        }

        void Group::setPhrases(int instrId, std::vector<unsigned>& phrases) {
            std::vector<unsigned> copy;

            for (unsigned& val : phrases) {
                copy.push_back(unsigned(val));
            }

            mSynthPhrases[instrId] = std::move(copy);
        }

        void Group::removePhrases(int instrId) {
            mSynthPhrases[instrId] = std::vector<unsigned>();
        }
}
#include "LoTide.hpp"
#include "Sequencer.hpp"

/*
  Groups are settings for how the song is played
  They are discrete
  They contain all instruments and information
  on what tracks those instruments play
```

```
  ------

  Chains are an implementation which allow for
  pathing through the graph in defined ways

*/

namespace lotide {

	LoTide::LoTide() : sequencer(90) {
	}

	Song& LoTide::addSong(std::string name, int tempo) {
		songs.emplace_back(Song(name, masterMixer, tempo));

		return songs[songs.size() - 1];
	}

	Song& LoTide::addSong(std::string name) {
		songs.emplace_back(Song(name, masterMixer, 90));

		return songs[songs.size() - 1];
	}

	void LoTide::setSong(std::string name) {
		for (Song& s : songs) {
			if (s.getName() == name) {
				sequencer.setSong(s);
			}
		}
	}

	void LoTide::setGroup(std::string name) {
		Song* activeSong = &sequencer.getSong();

		if (activeSong) {
			activeSong->setGroup(name);
		}
	}

	void LoTide::setNextGroup(std::string name) {
		Song* activeSong = &sequencer.getSong();
```

```
	if (activeSong) {
		activeSong->setNextGroup(name);
	}
}

void LoTide::play() {
	sequencer.start();
}

void LoTide::stop() {
	sequencer.stop();
}

// TODO make it so that it filters the name rather than getting a song
// TODO or make it so that it imports all the songs in song array
void LoTide::load(std::string name, std::string filePath) {
	Song song;
	std::ifstream ifs(filePath);

	if (ifs.good()) {
		cereal::JSONInputArchive archive_in(ifs);
		archive_in(song);
	} else {
		assert(false);
	}

	song.init(masterMixer);
	std::string songName = song.getName();
	sequencer.setTempo(song.getTempo());

	songs.emplace_back(std::move(song));
	setSong(songName);

}

void LoTide::save(std::string filePath) {
	Song& song = sequencer.getSong();

	std::ofstream ofs(filePath);

	if (ofs.good()) {
		cereal::JSONOutputArchive archive_out(ofs);
		archive_out(song);
```

```
		} else {
			assert(false);
		}
	}

	std::string LoTide::serializeJSON() {
		Song& song = sequencer.getSong();
		std::stringstream os;

		if (os.good()) {
			cereal::JSONOutputArchive archive_out(os);
			archive_out(CEREAL_NVP(song));
		} else {
			assert(false);
		}

		return os.str();
	}

	Song& LoTide::getActiveSong() {
		return sequencer.getSong();
	}

	void LoTide::setInstrumentPlay(std::string g2, int instrId) {
		Song& song = sequencer.getSong();

		if (song.getActiveGroup().getName() == "generated") {
			song.getActiveGroup().setPhrases(instrId,
song.getGroup(g2).getPhrases(instrId));
		}
		else {
			Group&& newGroup = Group(song.getActiveGroup());
			newGroup.setName("generated");

			Group& group2 = song.getGroup(g2);

			newGroup.setPhrases(instrId, group2.getPhrases(instrId));

			song.addGroup(std::move(newGroup));
			setGroup("generated");
		}
	}
```

```
	void LoTide::removeInstrument(int instrumentId) {
		Song& song = sequencer.getSong();

		if (song.getActiveGroup().getName() == "generated") {
			song.getActiveGroup().removePhrases(instrumentId);
		}
		else {
			Group&& newGroup = Group(song.getActiveGroup());
			newGroup.setName("generated");

			newGroup.removePhrases(instrumentId);

			song.addGroup(std::move(newGroup));
			setGroup("generated");
		}


	}
}
set(HEADER_LIST "${PROJECT_SOURCE_DIR}/include/LoTide.hpp")

SET(BUILD_SHARED_LIBS ON)

set(SOURCES
  Server.cpp
  LoTide.cpp
  Sequencer.cpp
  Note.cpp
  Song.cpp
  Phrase.cpp
  Group.cpp
  main.cpp
  LTSynth.cpp
  )

find_library(WSOCK32_LIBRARY wsock32)
find_library(WS2_32_LIBRARY ws2_32)

add_library(LoTide STATIC ${SOURCES} ${HEADER_LIST})
add_library(${LOCAL_PROJECT_NAMESPACE}::${LOCAL_PROJECT_NAME} ALIAS
LoTide)

target_include_directories(LoTide
```

```
  PUBLIC
  ${PROJECT_SOURCE_DIR}/include
)

if(WIN32)
  target_link_libraries(LoTide wsock32 ws2_32)
endif()

target_link_libraries(LoTide
  tsal::tsal
  cereal
  nlohmann_json::nlohmann_json
  )

add_executable(lotide-bin ${SOURCES})

target_include_directories(lotide-bin
  PUBLIC
  ${PROJECT_SOURCE_DIR}/include
)

if(WIN32)
  target_link_libraries(lotide-bin wsock32 ws2_32)
endif()

target_link_libraries(lotide-bin
  tsal::tsal
  cereal
  nlohmann_json::nlohmann_json
  )

set_target_properties(lotide-bin
  PROPERTIES
  OUTPUT_NAME lotide
  )
#include "Sequencer.hpp"


namespace lotide {

        Sequencer::Sequencer(unsigned bpm) : mBpm(bpm) {
                isPlaying = false;
                currentTime = -1;
```

```
		activeSong = NULL;
	}

	// Corresponds to when the songs should start to play
	// Timing events will now be relative to when this is called
	void Sequencer::start() {
		isPlaying = true;

		std::thread thread_obj(&Sequencer::tick, this);
		thread_obj.detach();
	}

	void Sequencer::start(unsigned time) {
		this->currentTime = time - 1;
		start();
	}

	void Sequencer::stop() {
		isPlaying = false;
		std::this_thread::sleep_for(std::chrono::seconds(1));
	}

	// Look in LoTide to find all notes which need to be added
	void Sequencer::findUpcomingNotes() {
		if (activeSong == NULL) {
			return;
		}

		std::unordered_map<unsigned, std::list<Note>> newUpcoming =
activeSong->getUpcoming(currentTime);
		for (std::pair<unsigned, std::list<Note>> newNotes : newUpcoming) {
			unsigned synthId = newNotes.first;

			for (Note newNote : newNotes.second) {
				addUpcoming(newNote, newNotes.first);
			}
		}
	}

	void Sequencer::processNotes() {
		std::map<unsigned, std::list<Note>> toBeRemoved;
```

```
		// check if notes need to be stopped
		for (std::pair<unsigned, std::list<Note>> playingNotes : playing) {
			std::vector<LTSynth*> synths = activeSong->getSynths();

			LTSynth* synth = synths[playingNotes.first];

			for (Note& playingNote : playingNotes.second) {
				double val = playingNote.getNote();

				if (playingNote.getDuration() <= abs(currentTime -
noteTimes.at(playingNote))) {
					//std::cout << "stopping " << val << std::endl;
					synth->stop(val);
					noteTimes.erase(playingNote);

toBeRemoved[playingNotes.first].push_back(playingNote);
				}
			}


		}

		for (std::pair<unsigned, std::list<Note>> remPair : toBeRemoved) {
			for (Note& n : remPair.second) {
				playing[remPair.first].remove(n);
			}
		}
	}

	void Sequencer::addUpcoming(Note note, unsigned synthId) {
		std::vector<LTSynth*> synths = activeSong->getSynths();

		if (playing.find(synthId) == playing.end()) {
			std::list<Note> synthNotes;
			synthNotes.push_back(note);

			playing.insert(std::pair<unsigned, std::list<Note>>(synthId,
std::move(synthNotes)));
		} else {
			playing[synthId].push_back(note);
		}

		noteTimes[note] = currentTime;
```

```
                LTSynth* synth = synths[synthId];
                synth->play(note.getNote(), note.getVelocity());

                //std::cout << "playing " << note.getNote() << std::endl;
        }

        void Sequencer::tick() {
                std::chrono::system_clock::time_point now =
std::chrono::system_clock::now();

                unsigned currentLength = activeSong->getLength();

                if (currentLength != 0) {
                        currentTime = (currentTime + 1) % currentLength;

                        if (currentTime == 0) {
                                clearAll();
                        }

                        processNotes();
                        findUpcomingNotes();

                }

                std::chrono::system_clock::time_point after =
std::chrono::system_clock::now();

                std::chrono::milliseconds time = max(std::chrono::milliseconds(0),
                        std::chrono::milliseconds(60000 / (mBpm * ppq))
                        - std::chrono::duration_cast<std::chrono::milliseconds>(after -
now));

                std::this_thread::sleep_for(time);

                if (isPlaying) {
                        tick();
                }
                else {
                        clearAll();
                }
        }
```

```
	void Sequencer::clearAll() {
		if (!activeSong) {
			return;
		}

		std::vector<LTSynth*> synths = activeSong->getSynths();

		for (std::pair<unsigned, std::list<Note>> notevector : playing) {
			LTSynth* synth = synths[notevector.first];

			for (double note = tsal::A0; note != tsal::Gs7; note++) {
				synth->stop(note);
			}
			synth->stop(tsal::Gs7);

			notevector.second.clear();
		}
	}

	void Sequencer::setSong(Song& s) {
		activeSong = &s;
		mBpm = s.getTempo();
	}

	void Sequencer::setCurrentTime(int pulse) {
		int length = activeSong->getLength();

		if (pulse < length) {
			currentTime = pulse;
		}
	}
}#include "Server.hpp"

namespace lotide {

	Server::Server(int port) : mPort(port) {
		std::cout << "Starting Server!" << std::endl;

#ifdef _WIN32
		if (WSAStartup(MAKEWORD(1,1), &wsaData) != 0) {
			std::cerr << "WSAStartup failed." << std::endl;
			exit(EXIT_FAILURE);
		}
```

```
#endif

#ifdef WIN32
		// Initialize Winsock
		WSAData wsaData;
		iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
		if (iResult != 0) {
			std::cout << "WSAStartup failed: " << iResult << std::endl;
			return;
		}
#endif
		// Create server
		listen_socket = socket(AF_INET, SOCK_STREAM, 0);
		if (listen_socket == -1) {
			std::cerr << "Cannot create socket." << std::endl;
			exit(EXIT_FAILURE);
		}

		// NOTE Clear address (basically like null assignment)
		memset(&address, 0, sizeof address);

		// TODO Consider AF_UNIX because it is just a file
		address.sin_family = AF_INET;
		address.sin_addr.s_addr = INADDR_ANY;
		address.sin_port = htons(mPort);

#ifdef WIN32
		// Initialize Winsock
		iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
		if (iResult != 0) {
			std::cout << "WSAStartup failed: " << iResult << std::endl;
			return;
		}
#endif
		// Bind to address
		if (bind(listen_socket, (struct sockaddr *)&address, sizeof(address)) == -1) {
			std::cerr << "Bind failed." << std::endl;
#ifdef _WIN32
			closesocket(listen_socket);
			WSACleanup();
#else
			close(listen_socket);
#endif
```

```
                std::cerr << "Exit failed." << std::endl;
                exit(EXIT_FAILURE);
        }

        // Listen to address
        if (listen(listen_socket, 10) == -1) {
                std::cerr << "Listen failed." << std::endl;
#ifdef _WIN32
                closesocket(listen_socket);
                WSACleanup();
#else
                close(listen_socket);
#endif
                std::cerr << "Exit failed." << std::endl;
                exit(EXIT_FAILURE);
        }
    }

    Server::~Server() {
        std::cout << "Server Closed." << std::endl;

#ifdef _WIN32
        closesocket(new_socket);
        WSACleanup();
#else
        close(new_socket);
#endif

    }

    void Server::parseExecute(std::string command, std::vector<std::string> params)
{
        if (command == "play") {
                // Need this to increment and refresh current length according to
added phrases
                lt.setGroup(activeInstanceGroup);
                lt.play();
        } else if (command == "stop") {
                lt.stop();
        } else if (command == "playNote") {
                Song& song = lt.getActiveSong();
                LTSynth& synth = song.getSynth(activeSynthId);
                synth.play(std::stod(params[0]), std::stod(params[1]));
```

```
                std::this_thread::sleep_for(std::chrono::microseconds(300000));
                synth.stop(std::stod(params[0]));
            } else if (command == "setAttack") {
                Song& song = lt.getActiveSong();
                LTSynth& synth = song.getSynth(activeSynthId);
                synth.getSynth().setParameter(tsal::PolySynth::ENV_ATTACK,
std::stod(params[0]));
            } else if (command == "setSustain") {
                Song& song = lt.getActiveSong();
                LTSynth& synth = song.getSynth(activeSynthId);
                synth.getSynth().setParameter(tsal::PolySynth::ENV_SUSTAIN,
std::stod(params[0]));
            } else if (command == "setDecay") {
                Song& song = lt.getActiveSong();
                LTSynth& synth = song.getSynth(activeSynthId);
                synth.getSynth().setParameter(tsal::PolySynth::ENV_DECAY,
std::stod(params[0]));
            } else if (command == "setRelease") {
                Song& song = lt.getActiveSong();
                LTSynth& synth = song.getSynth(activeSynthId);
                synth.getSynth().setParameter(tsal::PolySynth::ENV_RELEASE,
std::stod(params[0]));
            } else if (command == "setVolume") {
                Song& song = lt.getActiveSong();
                LTSynth& synth = song.getSynth(activeSynthId);
                synth.getSynth().setParameter(tsal::PolySynth::VOLUME,
std::stod(params[0]));
            } else if (command == "setModulationMode") {
                Song& song = lt.getActiveSong();
                LTSynth& synth = song.getSynth(activeSynthId);

synth.getSynth().setParameter(tsal::PolySynth::MODULATION_MODE,
std::stod(params[0]));
            } else if (command == "setOSC1Mode") {
                Song& song = lt.getActiveSong();
                LTSynth& synth = song.getSynth(activeSynthId);
                synth.getSynth().setParameter(tsal::PolySynth::OSC1_MODE,
std::stod(params[0]));
            } else if (command == "setOSC2Mode") {
                Song& song = lt.getActiveSong();
                LTSynth& synth = song.getSynth(activeSynthId);
                synth.getSynth().setParameter(tsal::PolySynth::OSC2_MODE,
std::stod(params[0]));
```

```
		} else if (command == "setOSC2Offset") {
			Song& song = lt.getActiveSong();
			LTSynth& synth = song.getSynth(activeSynthId);
			synth.getSynth().setParameter(tsal::PolySynth::OSC2_OFFSET,
std::stod(params[0]));
		} else if (command == "close") {
			std::cout << "Successfully Closed" << std::endl;
			exit(EXIT_SUCCESS);
		} else if (command == "addSong") {
			// FIXME No way to display active song
			lt.addSong(params[0]);
			lt.setSong(params[0]);
		} else if (command == "addSynth") {
			Song& song = lt.getActiveSong();
			LTSynth& synth = song.addSynth();
			activeSynthId = synth.getId();
		} else if (command == "addGroup") {
			Song& song = lt.getActiveSong();
			song.makeNewGroup(params[0]);
			lt.setGroup(params[0]);
			activeInstanceGroup = params[0];
		} else if (command == "addPhrase") {
			Song& song = lt.getActiveSong();
			Group& group = song.getActiveGroup();
			song.addPhrase(params[0]);
			group.addPhrase(activeSynthId, phraseIncrement);
			++phraseIncrement;
		} else if (command == "addNote") {
			Song& song = lt.getActiveSong();
			Phrase& phrase = song.getPhrase(activePhraseId);
			phrase.addNote(Note(std::stod(params[0]), std::stod(params[1]),
std::stoi(params[2]), std::stoi(params[3])));
		} else if (command == "removeNote") {
			Song& song = lt.getActiveSong();
			Phrase& phrase = song.getPhrase(activePhraseId);
			phrase.removeNote(std::stod(params[0]), std::stod(params[1]));
		} else if (command == "setActivePhrase") {
			activePhraseId = std::stoul(params[0]);
		} else if (command == "setActiveGroup") {
			activeInstanceGroup = params[0];
			lt.setGroup(params[0]);
		} else if (command == "setActiveSynth") {
			activeSynthId = std::stoul(params[0]);
```

```
		} else if (command == "setActiveSong") {
			lt.setSong(params[0]);
		} else if (command == "setLength") {
			Song& song = lt.getActiveSong();
			Phrase& phrase = song.getPhrase(activePhraseId);
			phrase.setLength(stoi(params[0]));
		} else if (command == "load") {
			lt.load(params[0], params[1]);
		} else if (command == "setInstrumentPlay") {	// replaces instrument
param[1]'s phrases in active group to whatever is in group param[0]
			lt.setInstrumentPlay(params[0], std::stoi(params[1]));
			activeInstanceGroup = "generated";
		} else if (command == "removeInstrument") {
			lt.removeInstrument(std::stoi(params[0]));
			activeInstanceGroup = "generated";
		} else if (command == "setTempo") {
			lt.setTempo(std::stoi(params[0]));
		} else {
			std::cout << "Wrong input" << std::endl;
		}
	}

	// TODO Filter Message size
	void Server::init() {
		for (;;) {
			// Open socket to connection [BLOCKING]
#ifdef WIN32
			// Initialize Winsock
			iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
			if (iResult != 0) {
				std::cout << "WSAStartup failed: " << iResult << std::endl;
				return;
			}
#endif

			client_socket = accept(listen_socket, NULL, NULL);
			// client_socket = accept(listen_socket, (struct sockaddr *)&address,
			//                                       (socklen_t *)&addrlen);

			if (client_socket == -1) {
#ifdef _WIN32
				std::cerr << "Accept failed with error: " << WSAGetLastError()
<< std::endl;
```

```
                        closesocket(listen_socket);
                        WSACleanup();
#else
                        close(listen_socket);
#endif
                        exit(EXIT_FAILURE);
                }

                // char recvbuf[4096];
                // std::vector<char> buffer(4096);

                std::vector<char> buf(4096); // create buffer with preallocated size

#ifdef WIN32
                iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
                if (iResult != 0) {
                        std::cout << "WSAStartup failed: " << iResult << std::endl;
                        return;
        }
#endif
                valread = recv(client_socket, &buf[0], buf.size(), 0);
                // valread = recv(client_socket, recvbuf, 4096, 0);
                buf.resize(valread);
                if (valread == -1) {
                        std::cerr << "Receive failed." << std::endl;
                        exit(EXIT_FAILURE);
                }

                // std::string receivedData(buffer.begin(), buffer.end());

                std::cout << buf.data() << std::endl;
                auto clientJSON = nlohmann::json::parse(buf.data());

                parseExecute(clientJSON["command"], clientJSON["parameters"]);
                std::cout << clientJSON["parameters"] << std::endl;

                songState = lt.serializeJSON();

#ifdef WIN32
                // Initialize Winsock
                iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
                if (iResult != 0) {
                        std::cout << "WSAStartup failed: " << iResult << std::endl;
```

```
                        return;
                }
#endif
                send(client_socket, songState.c_str(), songState.length(), 0);

#ifdef _WIN32
                if (shutdown(client_socket, SD_BOTH)) {
#else
                if (shutdown(client_socket, SHUT_RDWR) == -1) {
#endif
                        std::cerr << "Shutdown failed." << std::endl;
#ifdef _WIN32
                        closesocket(client_socket);
                        WSACleanup();
#else
                        close(client_socket);
#endif
                        std::cerr << "Close failed." << std::endl;
                        exit(EXIT_FAILURE);
                }

#ifdef _WIN32
                closesocket(client_socket);
                WSACleanup();
#else
                close(client_socket);
#endif
                }
        }
}
#include "Note.hpp"

namespace lotide {

unsigned Note::nextId = 0;

Note::Note() {
        note = 0;
        velocity = 0;
        startTime = 0;
        duration = 0;

        id = 0;
```

```
}

Note::Note(double n, double v, int s, int d) : note(n), velocity(v), startTime(s), duration(d),
id(nextId) {
	nextId++;
}

Note::Note(const Note& otherNote) {
	note = otherNote.getNote();
	velocity = otherNote.getVelocity();
	startTime = otherNote.getStartTime();
	duration = otherNote.getDuration();

	id = otherNote.id;
}

double Note::getNote() const {
	return note;
}

double Note::getVelocity() const {
	return velocity;
}

int Note::getStartTime() const {
	return startTime;
}

int Note::getDuration() const {
	return duration;
}

void Note::setNote(double n){
	note = n;
}

void Note::setVelocity(double v){
	velocity = v;
}

void Note::setStartTime(int t){
	startTime = t;
}
```

```
void Note::setDuration(int d){
	duration = d;
}


}
#include <iostream>

#include "LoTide.hpp"
#include "LTSynth.hpp"
#include "Group.hpp"
#include "Phrase.hpp"
#include "Sequencer.hpp"
#include "Note.hpp"

#include "Server.hpp"

using namespace lotide;

int main(int argc, char *argv[])
{
	lotide::LoTide lt;
	int q = Sequencer::ppq;


	Song& song = lt.addSong("Song1", 138);

	LTSynth& synth3 = song.addSynth();
	int synth3Id = synth3.getId();

	synth3.setParameter(tsal::PolySynth::ENV_ATTACK, 0.04);
	synth3.setParameter(tsal::PolySynth::ENV_SUSTAIN, .15);
	synth3.setParameter(tsal::PolySynth::ENV_DECAY, .2);
	synth3.setParameter(tsal::PolySynth::ENV_RELEASE, .2);

	synth3.setParameter(tsal::PolySynth::MODULATION_MODE, tsal::Oscillator::MIX);
	synth3.setParameter(tsal::PolySynth::OSC1_MODE, tsal::Oscillator::SQUARE);
	synth3.setParameter(tsal::PolySynth::OSC2_MODE, tsal::Oscillator::SAW);
	synth3.setParameter(tsal::PolySynth::OSC2_OFFSET, 0.3);
	//synth3.setParameter(tsal::PolySynth::VOLUME, .3);

	LTSynth& synth2 = song.addSynth();
```

```
int synth2Id = synth2.getId();

synth2.setParameter(tsal::PolySynth::ENV_ATTACK, 0.05);
synth2.setParameter(tsal::PolySynth::ENV_SUSTAIN, 0.4);
synth2.setParameter(tsal::PolySynth::ENV_DECAY, .25);
synth2.setParameter(tsal::PolySynth::ENV_RELEASE, .8);

synth2.setParameter(tsal::PolySynth::MODULATION_MODE, tsal::Oscillator::PM);
synth2.setParameter(tsal::PolySynth::OSC1_MODE, tsal::Oscillator::SINE);
synth2.setParameter(tsal::PolySynth::OSC2_MODE, tsal::Oscillator::SQUARE);
synth2.setParameter(tsal::PolySynth::OSC2_OFFSET, 0.4);


LTSynth& synth = song.addSynth();
int synth1Id = synth.getId();

synth.setParameter(tsal::PolySynth::ENV_ATTACK, .05);
synth.setParameter(tsal::PolySynth::ENV_SUSTAIN, .1);
synth.setParameter(tsal::PolySynth::ENV_DECAY, .6);
synth.setParameter(tsal::PolySynth::ENV_RELEASE, .8);

synth.setParameter(tsal::PolySynth::MODULATION_MODE, tsal::Oscillator::MIX);
synth.setParameter(tsal::PolySynth::OSC1_MODE, tsal::Oscillator::SINE);
synth.setParameter(tsal::PolySynth::OSC2_MODE, tsal::Oscillator::SAW);
synth.setParameter(tsal::PolySynth::OSC2_OFFSET, 0.5);

Phrase& guitar1 = song.addPhrase("LoruleGuitar1");
int guitar1Id = guitar1.getId();
guitar1.setLength(64);
guitar1.addNote(Note(tsal::Ds3, 44, 4, 4));
guitar1.addNote(Note(tsal::G3, 44, 4, 4));
guitar1.addNote(Note(tsal::Ds3, 44, 8, 4));
guitar1.addNote(Note(tsal::G3, 44, 8, 4));
guitar1.addNote(Note(tsal::Ds3, 44, 12, 2));
guitar1.addNote(Note(tsal::G3, 44, 12, 2));
guitar1.addNote(Note(tsal::Ds3, 44, 14, 2));
guitar1.addNote(Note(tsal::G3, 44, 14, 2));
guitar1.addNote(Note(tsal::Ds3, 44, 16, 4));
guitar1.addNote(Note(tsal::G3, 44, 16, 4));
guitar1.addNote(Note(tsal::F3, 44, 20, 4));
guitar1.addNote(Note(tsal::A3, 44, 20, 4));

guitar1.addNote(Note(tsal::G3, 44, 36, 4));
```

```
guitar1.addNote(Note(tsal::As3, 44, 36, 4));
guitar1.addNote(Note(tsal::G3, 44, 40, 4));
guitar1.addNote(Note(tsal::As3, 44, 40, 4));
guitar1.addNote(Note(tsal::G3, 44, 44, 2));
guitar1.addNote(Note(tsal::As3, 44, 44, 2));
guitar1.addNote(Note(tsal::G3, 44, 46, 2));
guitar1.addNote(Note(tsal::As3, 44, 46, 2));
guitar1.addNote(Note(tsal::G3, 44, 48, 4));
guitar1.addNote(Note(tsal::As3, 44, 48, 4));
guitar1.addNote(Note(tsal::A3, 44, 52, 4));
guitar1.addNote(Note(tsal::C4, 44, 52, 4));

Phrase& guitar2 = song.addPhrase("LoruleGuitar1");
int guitar2Id = guitar2.getId();
guitar2.setLength(128);
guitar2.addNote(Note(tsal::Ds3, 44, 4, 4));
guitar2.addNote(Note(tsal::G3, 44, 4, 4));
guitar2.addNote(Note(tsal::Ds3, 44, 8, 4));
guitar2.addNote(Note(tsal::G3, 44, 8, 4));
guitar2.addNote(Note(tsal::Ds3, 44, 12, 2));
guitar2.addNote(Note(tsal::G3, 44, 12, 2));
guitar2.addNote(Note(tsal::Ds3, 44, 14, 2));
guitar2.addNote(Note(tsal::G3, 44, 14, 2));
guitar2.addNote(Note(tsal::Ds3, 44, 16, 4));
guitar2.addNote(Note(tsal::G3, 44, 16, 4));
guitar2.addNote(Note(tsal::Ds3, 44, 20, 4));
guitar2.addNote(Note(tsal::G3, 44, 20, 4));

guitar2.addNote(Note(tsal::F3, 44, 36, 4));
guitar2.addNote(Note(tsal::A3, 44, 36, 4));
guitar2.addNote(Note(tsal::F3, 44, 40, 4));
guitar2.addNote(Note(tsal::A3, 44, 40, 4));
guitar2.addNote(Note(tsal::F3, 44, 44, 2));
guitar2.addNote(Note(tsal::A3, 44, 44, 2));
guitar2.addNote(Note(tsal::F3, 44, 46, 2));
guitar2.addNote(Note(tsal::A3, 44, 46, 2));
guitar2.addNote(Note(tsal::F3, 44, 48, 4));
guitar2.addNote(Note(tsal::A3, 44, 48, 4));
guitar2.addNote(Note(tsal::F3, 44, 52, 4));
guitar2.addNote(Note(tsal::A3, 44, 52, 4));

guitar2.addNote(Note(tsal::Gs3, 44, 68, 4));
guitar2.addNote(Note(tsal::Ds3, 44, 68, 4));
```

```
guitar2.addNote(Note(tsal::Gs3, 44, 72, 4));
guitar2.addNote(Note(tsal::Ds3, 44, 72, 4));
guitar2.addNote(Note(tsal::Gs3, 44, 76, 2));
guitar2.addNote(Note(tsal::Ds3, 44, 76, 2));
guitar2.addNote(Note(tsal::Gs3, 44, 78, 2));
guitar2.addNote(Note(tsal::Ds3, 44, 78, 2));
guitar2.addNote(Note(tsal::Gs3, 44, 80, 4));
guitar2.addNote(Note(tsal::Ds3, 44, 80, 4));
guitar2.addNote(Note(tsal::Gs3, 44, 84, 4));
guitar2.addNote(Note(tsal::Ds3, 44, 84, 4));

guitar2.addNote(Note(tsal::D3, 44, 64 + 36, 4));
guitar2.addNote(Note(tsal::F3, 44, 64 + 36, 4));
guitar2.addNote(Note(tsal::D3, 44, 64 + 40, 4));
guitar2.addNote(Note(tsal::F3, 44, 64 + 40, 4));
guitar2.addNote(Note(tsal::D3, 44, 64 + 44, 2));
guitar2.addNote(Note(tsal::F3, 44, 64 + 44, 2));
guitar2.addNote(Note(tsal::D3, 44, 64 + 46, 2));
guitar2.addNote(Note(tsal::F3, 44, 64 + 46, 2));
guitar2.addNote(Note(tsal::D3, 44, 64 + 48, 4));
guitar2.addNote(Note(tsal::F3, 44, 64 + 48, 4));
guitar2.addNote(Note(tsal::D3, 44, 64 + 52, 4));
guitar2.addNote(Note(tsal::F3, 44, 64 + 52, 4));

Phrase& bass1 = song.addPhrase("LoruleBass");
int bass1Id = bass1.getId();
bass1.setLength(64);
bass1.addNote(Note(tsal::C2, 33, 0, 6));
bass1.addNote(Note(tsal::C2, 33, 8, 6));
bass1.addNote(Note(tsal::C2, 33, 16, 6));
bass1.addNote(Note(tsal::C2, 33, 28, 4));

bass1.addNote(Note(tsal::C2, 33, 32, 6));
bass1.addNote(Note(tsal::C2, 33, 40, 6));
bass1.addNote(Note(tsal::C2, 33, 48, 6));
bass1.addNote(Note(tsal::C2, 33, 60, 4));

Phrase& bass2 = song.addPhrase("LoruleBass2");
int bass2Id = bass2.getId();
bass2.setLength(128);
bass2.addNote(Note(tsal::C2, 33, 0, 6));
bass2.addNote(Note(tsal::C2, 33, 8, 6));
bass2.addNote(Note(tsal::C2, 33, 16, 6));
```

```
bass2.addNote(Note(tsal::C2, 33, 28, 4));

bass2.addNote(Note(tsal::F2, 33, 32, 6));
bass2.addNote(Note(tsal::F2, 33, 40, 6));
bass2.addNote(Note(tsal::F2, 33, 48, 6));
bass2.addNote(Note(tsal::F2, 33, 60, 4));

bass2.addNote(Note(tsal::Gs2, 33, 64, 6));
bass2.addNote(Note(tsal::Gs2, 33, 72, 6));
bass2.addNote(Note(tsal::Gs2, 33, 80, 6));
bass2.addNote(Note(tsal::Gs2, 33, 92, 4));

bass2.addNote(Note(tsal::As2, 33, 96, 6));
bass2.addNote(Note(tsal::As2, 33, 104, 6));
bass2.addNote(Note(tsal::As2, 33, 112, 6));

Phrase& loruleSilence = song.addPhrase("LoruleMelodySilence");
int loruleSilenceId = loruleSilence.getId();
loruleSilence.setLength(64);

Phrase& loruleMelodyIntro = song.addPhrase("LoruleMelodySilence");
int loruleMelodyIntroId = loruleMelodyIntro.getId();
loruleMelodyIntro.setLength(64);
loruleMelodyIntro.addNote(Note(tsal::C4, 100, 62, 2));

Phrase& lorule = song.addPhrase("LoruleMelody1");
int loruleId = lorule.getId();
lorule.setLength(64);
lorule.addNote(Note(tsal::G4, 100, 0, 20));
lorule.addNote(Note(tsal::C4, 85, 20, 4));
lorule.addNote(Note(tsal::G4, 88, 24, 4));
lorule.addNote(Note(tsal::C5, 92, 28, 4));
lorule.addNote(Note(tsal::A4, 98, 32, 2));
lorule.addNote(Note(tsal::As4, 100, 34, 2));
lorule.addNote(Note(tsal::A4, 95, 36, 28));
lorule.addNote(Note(tsal::F4, 86, 62, 2));

Phrase& lorule2 = song.addPhrase("LoruleMelody2");
int lorule2Id = lorule2.getId();
lorule2.setLength(64);
lorule2.addNote(Note(tsal::G4, 100, 0, 8));
lorule2.addNote(Note(tsal::C4, 100, 8, 16));
lorule2.addNote(Note(tsal::F4, 100, 24, 8));
```

```
lorule2.addNote(Note(tsal::D4, 100, 32, 2));
lorule2.addNote(Note(tsal::Ds4, 100, 34, 2));
lorule2.addNote(Note(tsal::D4, 100, 36, 26));
lorule2.addNote(Note(tsal::As3, 100, 62, 2));

Phrase& lorule3 = song.addPhrase("LoruleMelody3");
int lorule3Id = lorule3.getId();
lorule3.setLength(64);
lorule3.addNote(Note(tsal::G5, 100, 0, 20));
lorule3.addNote(Note(tsal::C5, 85, 20, 4));
lorule3.addNote(Note(tsal::G5, 88, 24, 4));
lorule3.addNote(Note(tsal::C6, 92, 28, 4));
lorule3.addNote(Note(tsal::A5, 98, 32, 2));
lorule3.addNote(Note(tsal::As5, 100, 34, 2));
lorule3.addNote(Note(tsal::A5, 95, 36, 28));
lorule3.addNote(Note(tsal::F5, 86, 62, 2));

Phrase& lorule5 = song.addPhrase("LoruleMelody5");
int lorule5Id = lorule5.getId();
lorule5.setLength(64);
lorule5.addNote(Note(tsal::G5, 100, 0, 8));
lorule5.addNote(Note(tsal::C5, 100, 8, 16));
lorule5.addNote(Note(tsal::F5, 100, 24, 8));
lorule5.addNote(Note(tsal::D5, 100, 32, 2));
lorule5.addNote(Note(tsal::Ds5, 100, 34, 2));
lorule5.addNote(Note(tsal::D5, 100, 36, 26));
lorule5.addNote(Note(tsal::As4, 100, 62, 2));

Phrase& lorule4 = song.addPhrase("LoruleMelody4");
int lorule4Id = lorule4.getId();
lorule4.setLength(64);
lorule4.addNote(Note(tsal::C4, 100, 0, 32));
lorule4.addNote(Note(tsal::C5, 100, 62, 2));

Phrase& lorule6 = song.addPhrase("LoruleMelody6");
int lorule6Id = lorule6.getId();
lorule6.setLength(64);
lorule6.addNote(Note(tsal::C5, 100, 0, 32));

Group& g = song.makeNewGroup("part1");

g.addPhrase(synth1Id, loruleSilenceId);
g.addPhrase(synth1Id, loruleMelodyIntroId);
```

```
g.addPhrase(synth1Id, loruleId);
g.addPhrase(synth1Id, lorule2Id);
g.addPhrase(synth1Id, lorule4Id);
g.addPhrase(synth1Id, lorule3Id);
g.addPhrase(synth1Id, lorule5Id);
g.addPhrase(synth1Id, lorule6Id);

g.addPhrase(synth2Id, bass1Id);
g.addPhrase(synth2Id, bass1Id);
g.addPhrase(synth2Id, bass2Id);
g.addPhrase(synth2Id, bass1Id);
g.addPhrase(synth2Id, bass2Id);
g.addPhrase(synth2Id, bass1Id);

g.addPhrase(synth3Id, guitar1Id);
g.addPhrase(synth3Id, guitar1Id);
g.addPhrase(synth3Id, guitar2Id);
g.addPhrase(synth3Id, guitar1Id);
g.addPhrase(synth3Id, guitar2Id);
g.addPhrase(synth3Id, guitar1Id);

Group& g2 = song.makeNewGroup("part2");
Phrase& sunflower = song.addPhrase("Sunflower");
int sunflowerId = sunflower.getId();
sunflower.setLength(514);
// measure 1
sunflower.addNote(Note(tsal::G4, 100, 0, 32));
sunflower.addNote(Note(tsal::C5, 100, 0, 32));
sunflower.addNote(Note(tsal::E5, 100, 0, 32));
// measure 2
sunflower.addNote(Note(tsal::A4, 100, 32, 16));
sunflower.addNote(Note(tsal::D5, 100, 32, 16));
sunflower.addNote(Note(tsal::F5, 100, 32, 16));
sunflower.addNote(Note(tsal::G5, 100, 48, 12));
sunflower.addNote(Note(tsal::F5, 100, 60, 4));
// measure 3
sunflower.addNote(Note(tsal::A4, 100, 64, 16));
sunflower.addNote(Note(tsal::C5, 100, 64, 16));
sunflower.addNote(Note(tsal::E5, 100, 64, 16));
sunflower.addNote(Note(tsal::A4, 100, 80, 4));
sunflower.addNote(Note(tsal::B4, 100, 84, 4));
sunflower.addNote(Note(tsal::C5, 100, 88, 4));
sunflower.addNote(Note(tsal::D5, 100, 92, 4));
```

```
// measure 4
sunflower.addNote(Note(tsal::Gs4, 100, 96, 8));
sunflower.addNote(Note(tsal::C5, 100, 96, 8));
sunflower.addNote(Note(tsal::G5, 100, 96, 8));
sunflower.addNote(Note(tsal::F5, 100, 104, 12));
sunflower.addNote(Note(tsal::C5, 100, 116, 8));
sunflower.addNote(Note(tsal::D5, 100, 124, 4));
// measure 5
sunflower.addNote(Note(tsal::A4, 100, 128, 16));
sunflower.addNote(Note(tsal::C5, 100, 128, 16));
sunflower.addNote(Note(tsal::E5, 100, 128, 16));
sunflower.addNote(Note(tsal::A5, 100, 144, 16));
// measure 5
sunflower.addNote(Note(tsal::E5, 100, 160, 16));
sunflower.addNote(Note(tsal::Gs5, 100, 160, 16));
sunflower.addNote(Note(tsal::B5, 100, 160, 16));
sunflower.addNote(Note(tsal::C6, 100, 176, 8));
sunflower.addNote(Note(tsal::D6, 100, 184, 8));
// measure 6
sunflower.addNote(Note(tsal::B5, 100, 192, 24));
sunflower.addNote(Note(tsal::C6, 100, 192, 24));
sunflower.addNote(Note(tsal::E6, 100, 192, 24));
sunflower.addNote(Note(tsal::C6, 100, 216, 8));
// measure 7
sunflower.addNote(Note(tsal::C5, 100, 224, 32));
sunflower.addNote(Note(tsal::E5, 100, 224, 32));
sunflower.addNote(Note(tsal::A5, 100, 224, 32));
// measure 8
sunflower.addNote(Note(tsal::A4, 100, 256, 24));
sunflower.addNote(Note(tsal::C5, 100, 256, 24));
sunflower.addNote(Note(tsal::F5, 100, 256, 24));
sunflower.addNote(Note(tsal::A5, 100, 280, 8));
// measure 9
sunflower.addNote(Note(tsal::D5, 100, 288, 8));
sunflower.addNote(Note(tsal::F5, 100, 288, 8));
sunflower.addNote(Note(tsal::A5, 100, 288, 8));
sunflower.addNote(Note(tsal::G5, 100, 296, 8));
sunflower.addNote(Note(tsal::B4, 100, 304, 8));
sunflower.addNote(Note(tsal::D5, 100, 304, 8));
sunflower.addNote(Note(tsal::G5, 100, 304, 8));
sunflower.addNote(Note(tsal::F5, 100, 312, 8));
// measure 10
sunflower.addNote(Note(tsal::B4, 100, 320, 24));
```

```
sunflower.addNote(Note(tsal::D5, 100, 320, 24));
sunflower.addNote(Note(tsal::G5, 100, 320, 24));
sunflower.addNote(Note(tsal::D5, 100, 344, 8));
// measure 11
sunflower.addNote(Note(tsal::G4, 100, 352, 16));
sunflower.addNote(Note(tsal::A4, 100, 352, 16));
sunflower.addNote(Note(tsal::Cs5, 100, 352, 16));
sunflower.addNote(Note(tsal::D5, 100, 368, 8));
sunflower.addNote(Note(tsal::E5, 100, 376, 8));
// measure 12
sunflower.addNote(Note(tsal::A4, 100, 384, 32));
sunflower.addNote(Note(tsal::D5, 100, 384, 32));
sunflower.addNote(Note(tsal::F5, 100, 384, 32));
// measure 13
sunflower.addNote(Note(tsal::Gs4, 100, 416, 32));
sunflower.addNote(Note(tsal::Cs5, 100, 416, 32));
sunflower.addNote(Note(tsal::F5, 100, 416, 32));
// loop point
// measure 14 outro
sunflower.addNote(Note(tsal::G4, 100, 448, 16));
sunflower.addNote(Note(tsal::C5, 100, 448, 16));
sunflower.addNote(Note(tsal::E5, 100, 448, 16));
sunflower.addNote(Note(tsal::G4, 100, 468, 4));
sunflower.addNote(Note(tsal::B4, 100, 472, 4));
sunflower.addNote(Note(tsal::C5, 100, 476, 4));
// measure 15 outro
sunflower.addNote(Note(tsal::D5, 100, 480, 6));
sunflower.addNote(Note(tsal::E5, 100, 486, 28));


Phrase& sunflowerBass = song.addPhrase("SunflowerBass");
int sunflowerBassId = sunflowerBass.getId();
sunflowerBass.setLength(514);
// measure 1
sunflowerBass.addNote(Note(tsal::C4, 100, 0, 32));
// measure 2
sunflowerBass.addNote(Note(tsal::As3, 100, 32, 32));
// measure 3
sunflowerBass.addNote(Note(tsal::A3, 100, 64, 32));
// measure 4
sunflowerBass.addNote(Note(tsal::Gs3, 100, 96, 32));
// measure 5
sunflowerBass.addNote(Note(tsal::A3, 100, 128, 32));
```

```
// measure 6
sunflowerBass.addNote(Note(tsal::Gs3, 100, 160, 32));
// measure 7
sunflowerBass.addNote(Note(tsal::G3, 100, 192, 32));
// measure 8
sunflowerBass.addNote(Note(tsal::Fs4, 100, 224, 32));
// measure 9
sunflowerBass.addNote(Note(tsal::D5, 100, 256, 32));
// measure 10
sunflowerBass.addNote(Note(tsal::G5, 100, 288, 16));
sunflowerBass.addNote(Note(tsal::F5, 100, 304, 16));
// measure 11
sunflowerBass.addNote(Note(tsal::E5, 100, 320, 32));
// measure 12
sunflowerBass.addNote(Note(tsal::A4, 100, 352, 32));
// measure 13
sunflowerBass.addNote(Note(tsal::D5, 100, 384, 32));
// measure 14
sunflowerBass.addNote(Note(tsal::Cs4, 100, 416, 32));
// measure 15
sunflowerBass.addNote(Note(tsal::C4, 100, 448, 32));
sunflowerBass.addNote(Note(tsal::C4, 100, 480, 32));

g2.addPhrase(synth1Id, sunflowerId);
g2.addPhrase(synth2Id, sunflowerBassId);

Group& g3 = song.makeNewGroup("part3");
Phrase& kkRestMeasure = song.addPhrase("KKRestMeasure");
int kkRestMeasureId = kkRestMeasure.getId();
kkRestMeasure.setLength(128);

Phrase& kkHouseIntro = song.addPhrase("KKHouseIntro");
int kkHouseIntroId = kkHouseIntro.getId();
kkHouseIntro.setLength(128);
kkHouseIntro.addNote(Note(tsal::D3, 44, 0, 6));
kkHouseIntro.addNote(Note(tsal::A3, 44, 6, 6));
kkHouseIntro.addNote(Note(tsal::D4, 44, 12, 8));
kkHouseIntro.addNote(Note(tsal::D3, 44, 20, 4));
kkHouseIntro.addNote(Note(tsal::A3, 44, 24, 4));
kkHouseIntro.addNote(Note(tsal::D3, 44, 28, 2));
kkHouseIntro.addNote(Note(tsal::Ds3, 44, 30, 2));

kkHouseIntro.addNote(Note(tsal::E3, 44, 32, 6));
```

```
kkHouseIntro.addNote(Note(tsal::B3, 44, 38, 6));
kkHouseIntro.addNote(Note(tsal::E4, 44, 44, 8));
kkHouseIntro.addNote(Note(tsal::E3, 44, 52, 4));
kkHouseIntro.addNote(Note(tsal::B3, 44, 56, 4));
kkHouseIntro.addNote(Note(tsal::G2, 44, 60, 2));
kkHouseIntro.addNote(Note(tsal::Gs2, 44, 62, 2));

kkHouseIntro.addNote(Note(tsal::A2, 44, 64, 6));
kkHouseIntro.addNote(Note(tsal::E3, 44, 70, 6));
kkHouseIntro.addNote(Note(tsal::A3, 44, 76, 8));
kkHouseIntro.addNote(Note(tsal::A2, 44, 84, 4));
kkHouseIntro.addNote(Note(tsal::E3, 44, 88, 4));
kkHouseIntro.addNote(Note(tsal::G2, 44, 92, 2));
kkHouseIntro.addNote(Note(tsal::Gs2, 44, 94, 2));

kkHouseIntro.addNote(Note(tsal::A2, 44, 96, 6));
kkHouseIntro.addNote(Note(tsal::E3, 44, 102, 6));
kkHouseIntro.addNote(Note(tsal::As3, 44, 106, 2));
kkHouseIntro.addNote(Note(tsal::B3, 44, 108, 8));
kkHouseIntro.addNote(Note(tsal::A3, 44, 116, 4));
kkHouseIntro.addNote(Note(tsal::E3, 44, 120, 4));
kkHouseIntro.addNote(Note(tsal::Ds3, 44, 124, 4));

Phrase& kkHouseBass = song.addPhrase("KKHouseBass");
int kkHouseBassId = kkHouseBass.getId();
kkHouseBass.setLength(128);
kkHouseBass.addNote(Note(tsal::D2, 44, 0, 6));
kkHouseBass.addNote(Note(tsal::A2, 44, 6, 6));
kkHouseBass.addNote(Note(tsal::D3, 44, 12, 8));
kkHouseBass.addNote(Note(tsal::D2, 44, 20, 4));
kkHouseBass.addNote(Note(tsal::A2, 44, 24, 4));
kkHouseBass.addNote(Note(tsal::D2, 44, 28, 2));
kkHouseBass.addNote(Note(tsal::Ds3, 44, 30, 2));

kkHouseBass.addNote(Note(tsal::E2, 44, 32, 6));
kkHouseBass.addNote(Note(tsal::B2, 44, 38, 6));
kkHouseBass.addNote(Note(tsal::E3, 44, 44, 8));
kkHouseBass.addNote(Note(tsal::E2, 44, 52, 4));
kkHouseBass.addNote(Note(tsal::B2, 44, 56, 4));
kkHouseBass.addNote(Note(tsal::G1, 44, 60, 2));
kkHouseBass.addNote(Note(tsal::Gs1, 44, 62, 2));

kkHouseBass.addNote(Note(tsal::A1, 44, 64, 6));
```

```
kkHouseBass.addNote(Note(tsal::E2, 44, 70, 6));
kkHouseBass.addNote(Note(tsal::A2, 44, 76, 8));
kkHouseBass.addNote(Note(tsal::A1, 44, 84, 4));
kkHouseBass.addNote(Note(tsal::E2, 44, 88, 4));
kkHouseBass.addNote(Note(tsal::G1, 44, 92, 2));
kkHouseBass.addNote(Note(tsal::Gs1, 44, 94, 2));

kkHouseBass.addNote(Note(tsal::A1, 44, 96, 6));
kkHouseBass.addNote(Note(tsal::E2, 44, 102, 6));
kkHouseBass.addNote(Note(tsal::As2, 44, 106, 2));
kkHouseBass.addNote(Note(tsal::B2, 44, 108, 8));
kkHouseBass.addNote(Note(tsal::A2, 44, 116, 4));
kkHouseBass.addNote(Note(tsal::E2, 44, 120, 4));
kkHouseBass.addNote(Note(tsal::Ds2, 44, 124, 4));

Phrase& kkHouseRhythm = song.addPhrase("KKHouseRhythm");
int kkHouseRhythmId = kkHouseRhythm.getId();
kkHouseRhythm.setLength(128);

kkHouseRhythm.addNote(Note(tsal::D3, 44, 0, 8));

kkHouseRhythm.addNote(Note(tsal::A3, 44, 8, 8));
kkHouseRhythm.addNote(Note(tsal::C4, 44, 8, 8));
kkHouseRhythm.addNote(Note(tsal::E4, 44, 8, 8));

kkHouseRhythm.addNote(Note(tsal::A3, 44, 16, 4));
kkHouseRhythm.addNote(Note(tsal::C4, 44, 16, 4));
kkHouseRhythm.addNote(Note(tsal::E4, 44, 16, 4));

kkHouseRhythm.addNote(Note(tsal::F3, 44, 20, 2));

kkHouseRhythm.addNote(Note(tsal::A3, 44, 22, 4));
kkHouseRhythm.addNote(Note(tsal::C4, 44, 22, 4));
kkHouseRhythm.addNote(Note(tsal::E4, 44, 22, 4));

kkHouseRhythm.addNote(Note(tsal::F3, 44, 26, 2));

kkHouseRhythm.addNote(Note(tsal::A3, 44, 28, 4));
kkHouseRhythm.addNote(Note(tsal::C4, 44, 28, 4));
kkHouseRhythm.addNote(Note(tsal::E4, 44, 28, 4));

kkHouseRhythm.addNote(Note(tsal::E3, 44, 32, 8));
```

```
kkHouseRhythm.addNote(Note(tsal::B3, 44, 40, 8));
kkHouseRhythm.addNote(Note(tsal::D4, 44, 40, 8));
kkHouseRhythm.addNote(Note(tsal::Fs4, 44, 40, 8));

kkHouseRhythm.addNote(Note(tsal::B3, 44, 48, 4));
kkHouseRhythm.addNote(Note(tsal::D4, 44, 48, 4));
kkHouseRhythm.addNote(Note(tsal::Fs4, 44, 48, 4));

kkHouseRhythm.addNote(Note(tsal::G3, 44, 52, 2));

kkHouseRhythm.addNote(Note(tsal::B3, 44, 54, 4));
kkHouseRhythm.addNote(Note(tsal::D4, 44, 54, 4));
kkHouseRhythm.addNote(Note(tsal::Fs4, 44, 54, 4));

kkHouseRhythm.addNote(Note(tsal::G3, 44, 58, 2));

kkHouseRhythm.addNote(Note(tsal::B3, 44, 60, 4));
kkHouseRhythm.addNote(Note(tsal::D4, 44, 60, 4));
kkHouseRhythm.addNote(Note(tsal::Fs4, 44, 60, 4));

kkHouseRhythm.addNote(Note(tsal::A2, 44, 64, 8));

kkHouseRhythm.addNote(Note(tsal::E3, 44, 72, 8));
kkHouseRhythm.addNote(Note(tsal::Gs3, 44, 72, 8));
kkHouseRhythm.addNote(Note(tsal::Cs4, 44, 72, 8));

kkHouseRhythm.addNote(Note(tsal::E3, 44, 80, 4));
kkHouseRhythm.addNote(Note(tsal::Gs3, 44, 80, 4));
kkHouseRhythm.addNote(Note(tsal::Cs4, 44, 80, 4));

kkHouseRhythm.addNote(Note(tsal::E3, 44, 84, 2));

kkHouseRhythm.addNote(Note(tsal::E3, 44, 86, 4));
kkHouseRhythm.addNote(Note(tsal::Gs3, 44, 86, 4));
kkHouseRhythm.addNote(Note(tsal::Cs4, 44, 86, 4));

kkHouseRhythm.addNote(Note(tsal::E3, 44, 90, 2));

kkHouseRhythm.addNote(Note(tsal::E3, 44, 92, 4));
kkHouseRhythm.addNote(Note(tsal::Gs3, 44, 92, 4));
kkHouseRhythm.addNote(Note(tsal::Cs4, 44, 92, 4));

kkHouseRhythm.addNote(Note(tsal::A2, 44, 96, 4));
```

```
kkHouseRhythm.addNote(Note(tsal::Gs3, 44, 100, 4));
kkHouseRhythm.addNote(Note(tsal::Cs4, 44, 100, 4));
kkHouseRhythm.addNote(Note(tsal::E4, 44, 100, 4));

kkHouseRhythm.addNote(Note(tsal::E3, 44, 104, 2));

kkHouseRhythm.addNote(Note(tsal::Gs3, 44, 106, 4));
kkHouseRhythm.addNote(Note(tsal::Cs4, 44, 106, 4));
kkHouseRhythm.addNote(Note(tsal::E4, 44, 106, 4));

kkHouseRhythm.addNote(Note(tsal::E3, 44, 110, 2));

kkHouseRhythm.addNote(Note(tsal::G3, 44, 112, 4));
kkHouseRhythm.addNote(Note(tsal::As3, 44, 112, 4));
kkHouseRhythm.addNote(Note(tsal::Cs4, 44, 112, 4));
kkHouseRhythm.addNote(Note(tsal::F4, 44, 112, 4));

kkHouseRhythm.addNote(Note(tsal::Ds3, 44, 116, 2));

kkHouseRhythm.addNote(Note(tsal::Ds3, 44, 118, 4));
kkHouseRhythm.addNote(Note(tsal::G3, 44, 118, 4));
kkHouseRhythm.addNote(Note(tsal::As3, 44, 118, 4));
kkHouseRhythm.addNote(Note(tsal::Cs4, 44, 118, 4));
kkHouseRhythm.addNote(Note(tsal::F4, 44, 118, 4));

kkHouseRhythm.addNote(Note(tsal::Ds3, 44, 122, 2));

kkHouseRhythm.addNote(Note(tsal::Ds3, 44, 124, 4));
kkHouseRhythm.addNote(Note(tsal::G3, 44, 124, 4));
kkHouseRhythm.addNote(Note(tsal::As3, 44, 124, 4));
kkHouseRhythm.addNote(Note(tsal::Cs4, 44, 124, 4));
kkHouseRhythm.addNote(Note(tsal::F4, 44, 124, 4));

Phrase& kkHouseMelody1 = song.addPhrase("KKHouseMelody1");
int kkHouseMelody1Id = kkHouseMelody1.getId();
kkHouseMelody1.setLength(128);
kkHouseMelody1.addNote(Note(tsal::E5, 44, 0, 4));
kkHouseMelody1.addNote(Note(tsal::A4, 44, 4, 4));
kkHouseMelody1.addNote(Note(tsal::C5, 44, 8, 2));
kkHouseMelody1.addNote(Note(tsal::D5, 44, 10, 4));
kkHouseMelody1.addNote(Note(tsal::E5, 44, 14, 10));
```

```
kkHouseMelody1.addNote(Note(tsal::A4, 44, 36, 4));
kkHouseMelody1.addNote(Note(tsal::C5, 44, 40, 2));
kkHouseMelody1.addNote(Note(tsal::G5, 44, 42, 4));
kkHouseMelody1.addNote(Note(tsal::E5, 44, 46, 10));

kkHouseMelody1.addNote(Note(tsal::E5, 44, 68, 4));
kkHouseMelody1.addNote(Note(tsal::G5, 44, 72, 2));
kkHouseMelody1.addNote(Note(tsal::A5, 44, 74, 4));
kkHouseMelody1.addNote(Note(tsal::E5, 44, 78, 18));

Phrase& kkHouseMelody2 = song.addPhrase("KKHouseMelody2");
int kkHouseMelody2Id = kkHouseMelody2.getId();
kkHouseMelody2.setLength(128);
kkHouseMelody2.addNote(Note(tsal::E5, 44, 0,  4));
kkHouseMelody2.addNote(Note(tsal::A4, 44, 4, 4));
kkHouseMelody2.addNote(Note(tsal::C5, 44, 8, 2));
kkHouseMelody2.addNote(Note(tsal::D5, 44, 10, 4));
kkHouseMelody2.addNote(Note(tsal::E5, 44, 14, 10));

kkHouseMelody2.addNote(Note(tsal::A4, 44, 36, 4));
kkHouseMelody2.addNote(Note(tsal::C5, 44, 40, 2));
kkHouseMelody2.addNote(Note(tsal::D5, 44, 42, 4));
kkHouseMelody2.addNote(Note(tsal::E5, 44, 46, 10));

kkHouseMelody2.addNote(Note(tsal::A4, 44, 68, 4));
kkHouseMelody2.addNote(Note(tsal::A4, 44, 72, 2));
kkHouseMelody2.addNote(Note(tsal::G5, 44, 74, 4));
kkHouseMelody2.addNote(Note(tsal::E5, 44, 78, 18));

Phrase& kkHouseMelody3 = song.addPhrase("KKHouseMelody3");
int kkHouseMelody3Id = kkHouseMelody3.getId();
kkHouseMelody3.setLength(128);
kkHouseMelody3.addNote(Note(tsal::E5, 44, 0, 4));
kkHouseMelody3.addNote(Note(tsal::A4, 44, 4, 4));
kkHouseMelody3.addNote(Note(tsal::C5, 44, 8, 2));
kkHouseMelody3.addNote(Note(tsal::D5, 44, 10, 4));
kkHouseMelody3.addNote(Note(tsal::E5, 44, 14, 10));

kkHouseMelody3.addNote(Note(tsal::A4, 44, 36, 4));
kkHouseMelody3.addNote(Note(tsal::C5, 44, 40, 2));
kkHouseMelody3.addNote(Note(tsal::D5, 44, 42, 4));
kkHouseMelody3.addNote(Note(tsal::Ds5, 44, 46, 6));
kkHouseMelody3.addNote(Note(tsal::D5, 44, 52, 4));
```

```
        kkHouseMelody3.addNote(Note(tsal::C5, 44, 56, 4));
        kkHouseMelody3.addNote(Note(tsal::A4, 44, 60, 4));

        g3.addPhrase(synth1Id, kkHouseIntroId);
        g3.addPhrase(synth2Id, kkRestMeasureId);
        g3.addPhrase(synth3Id, kkRestMeasureId);

        g3.addPhrase(synth1Id, kkHouseMelody1Id);
        g3.addPhrase(synth1Id, kkHouseMelody2Id);
        g3.addPhrase(synth1Id, kkHouseMelody3Id);

        g3.addPhrase(synth2Id, kkHouseBassId);
        g3.addPhrase(synth2Id, kkHouseBassId);
        g3.addPhrase(synth2Id, kkHouseBassId);

        g3.addPhrase(synth3Id, kkHouseRhythmId);
        g3.addPhrase(synth3Id, kkHouseRhythmId);
        g3.addPhrase(synth3Id, kkHouseRhythmId);

        lt.setSong("Song1");
        lt.setGroup("part3");

        char input;

        lt.save("temp.lot");


        LoTide lt2;
        lt2.load("Song1", "temp.lot");
        lt2.setGroup("part2");

        std::cout << "Press <enter> to begin:" << std::endl;
        std::cin.get(input);

        lt2.play();


        std::cout << "Press <enter> to quit:" << std::endl;
        std::cin.get(input);

        lt2.stop();
}
#include "Phrase.hpp"
```

```
namespace lotide {
	Phrase::Phrase() {}

	Phrase::Phrase(std::string name, unsigned id) : mName(name), mId(id) {
		mLength = 0;
	}

	Phrase::Phrase(std::string newName, unsigned id, Phrase& otherPhrase) :
mName(newName), mId(id) {
		for (Note note : otherPhrase.getNotes()) {
			mNotes.push_back(Note(note));
		}
	}

	void Phrase::setLength(int newLength) {
		if (newLength >= mLength) {
			mLength = newLength;
		}
	}

	void Phrase::addNote(Note&& n) {
		mNotes.push_back(n);

		for (auto& note : mNotes) {
			unsigned noteLength = note.getStartTime() + note.getDuration();

			setLength(noteLength);
		}
	}

	void Phrase::removeNote(double myNote, int startTime) {
		for (auto& note : mNotes) {
			if (note.getNote() == myNote && note.getStartTime() == startTime) {
				mNotes.erase(std::remove(mNotes.begin(), mNotes.end(),
note), mNotes.end());

				unsigned noteLength = note.getStartTime() +
note.getDuration();

				if (mLength == noteLength) {
					mLength -= note.getDuration();
				}
```

```
                }
            }
        }

}
#ifndef LTSYNTH_HPP
#define LTSYNTH_HPP

#include <cereal/access.hpp>
#include <cereal/archives/xml.hpp>
#include <cereal/archives/json.hpp>

#include "tsal.hpp"

namespace lotide {
    class LTSynth {

    public:
        LTSynth();
        LTSynth(unsigned id);
        unsigned getId() { return mId; }
        void play(double note, double velocity) { mSynth.play(note, velocity); }
        void stop(double note) { mSynth.stop(note); }

        tsal::PolySynth& getSynth() { return mSynth; }
        void setParameter(tsal::PolySynth::Parameters param, double val) {
            mParams.push_back(param);
            mVals.push_back(val);
            mSynth.setParameter(param, val);
        }
    private:
        friend class cereal::access;

        template<class Archive>
        void save(Archive & ar) const
        {
            // ar & mSynth;
            ar(
                cereal::make_nvp("id", mId),
                cereal::make_nvp("vals", mVals),
                cereal::make_nvp("params", mParams)
            );
        }
```

```
		template<class Archive>
		void load(Archive& ar)
		{
			// ar & mSynth;
			ar(
				cereal::make_nvp("id", mId),
				cereal::make_nvp("vals", mVals),
				cereal::make_nvp("params", mParams)
			);

			for (int i = 0; i < mParams.size(); i++) {
				mSynth.setParameter(mParams[i], mVals[i]);
			}
		}

		tsal::PolySynth mSynth;
		unsigned mId;

		std::vector<tsal::PolySynth::Parameters> mParams;
		std::vector<double> mVals;
	};
}

#endif
#ifndef SONG_HPP
#define SONG_HPP

#include <cereal/access.hpp>
#include <cereal/types/vector.hpp>
#include <cereal/types/string.hpp>
#include <cereal/types/memory.hpp>
#include <cereal/types/unordered_map.hpp>
#include <cereal/archives/xml.hpp>
#include <cereal/archives/json.hpp>

#include <string>
#include <list>
#include <memory>
#include "tsal.hpp"

#include "Phrase.hpp"
#include "Group.hpp"
```

```
#include "LTSynth.hpp"

namespace lotide {

	class Song {

	public:
		Song();
		Song(std::string name, tsal::Mixer& m, int tempo);
		Song(Song&& other) noexcept;
		/*Song(Song& other, tsal::Mixer& m);*/
		LTSynth& addSynth();
		std::vector<unsigned> getSynthIds();
		std::unordered_map<unsigned, std::list<Note>> getUpcoming(unsigned
time);
		std::vector<LTSynth*> getSynths();
		Phrase& addPhrase(std::string name);
		unsigned getLength() { return mCurrentLength; }
		Group& makeNewGroup(std::string groupName);
		std::string getName() { return mName; }
		void addGroup(Group&& g) {
			if (g.getName() == "generated") {
				int found = -1;
				for (int i = 0; i < groups.size(); i++) {
					if (groups[i].getName() == "generated") {
						groups[i] = g;
						found = i;
						break;
					}
				}

				if (found == -1) {
					groups.push_back(g);
				}
			}
		}
		void setGroup(std::string name);
		void setGroup(Group& g);
		void setNextGroup(std::string name);
		void setNextGroup(Group& g);
		Phrase& getPhrase(unsigned phraseId);
		LTSynth& getSynth(unsigned synthId);
		Group& getActiveGroup();
```

```
        Group& getGroup(std::string groupName) {
            for (Group& g : groups) {
                if (g.getName() == groupName) {
                    return g;
                }
            }
        }
        Group& getGroup(int id) {
            return groups[id];
        }
        void init(tsal::Mixer& m) {
            mMixer = &m;

            for (auto const& s : mSynths) {
                m.add(s->getSynth());
            }
        }
        void setTempo(int bpm) {
            mTempo = bpm;
        }
        int getTempo() { return mTempo; }
    private:
        friend class cereal::access;
        template<class Archive>
        void serialize(Archive & ar)
        {
            ar(cereal::make_nvp("name", mName),
                cereal::make_nvp("synths", mSynths),
                cereal::make_nvp("phrases", mPhrases),
                cereal::make_nvp("groups", groups),
                cereal::make_nvp("active_group", activeGroupName),
                cereal::make_nvp("current_length", mCurrentLength),
                cereal::make_nvp("next_unique_synth_id",
mNextUniqueSynthId),
                cereal::make_nvp("next_unique_phrase_id",
mNextUniquePhraseId),
                cereal::make_nvp("tempo", mTempo)
            );
        }

        std::string mName;
        tsal::Mixer* mMixer;
        std::vector<std::unique_ptr<LTSynth>> mSynths;
```

```
            //std::unordered_map<unsigned, std::vector<Phrase>> mSynthPhrases;
            std::vector<Phrase> mPhrases;
            std::vector<Group> groups;
            Group* activeGroup;
            Group* nextGroup = NULL;
            unsigned mCurrentLength;
            unsigned mNextUniqueSynthId = 0;
            unsigned mNextUniquePhraseId = 0;
            std::string activeGroupName;
            int mTempo;
        };

}

#endif
#ifndef GROUP_HPP
#define GROUP_HPP

#include <cereal/access.hpp>
#include <cereal/types/vector.hpp>
#include <cereal/types/unordered_map.hpp>
#include <cereal/types/string.hpp>
#include <cereal/archives/xml.hpp>
#include <cereal/archives/json.hpp>

#include "tsal.hpp"
#include "Phrase.hpp"

namespace lotide {

        class Group {

        public:
            Group();
            Group(std::string name, std::vector<unsigned> songSynths);
            void addPhrase(unsigned synthId, unsigned phraseId);
            std::vector<unsigned>& getPhrases(unsigned synthId);
            void addSynth(unsigned synthId);
            std::string getName() { return mName; }
            void addNextGroup(int groupId);
            int getNextGroup() { return nextGroup; }
            void setPhrases(int instrId, std::vector<unsigned>& phrases);
            void removePhrases(int instrId);
```

```
		void setName(std::string n) {
			mName = n;
		}
	private:
		friend class cereal::access;
		template<class Archive>
		void serialize(Archive & ar)
		{
			ar(mName, mSynthPhrases);
		}

		std::string mName;
		std::unordered_map<unsigned, std::vector<unsigned>> mSynthPhrases;
		int nextGroup = NULL;
	};

}

#endif
#ifndef LOTIDE_HPP
#define LOTIDE_HPP

#include <cereal/archives/xml.hpp>
#include <cereal/archives/json.hpp>
#include <cereal/types/concepts/pair_associative_container.hpp>
#include <cereal/types/unordered_map.hpp>
#include <cereal/types/memory.hpp>
#include <cereal/types/vector.hpp>

#include <vector>
#include <string>
#include <iostream>
#include <fstream>
#include <unordered_map>

#ifndef TSF_IMPLEMENTATION
#define TSF_IMPLEMENTATION
#include "tsal.hpp"
#include "Sequencer.hpp"
#include "Note.hpp"
#include "Song.hpp"

namespace lotide {
```

```
	class LoTide {
	public:
		LoTide();
		void play();
		void stop();
		void load(std::string name, std::string filePath);
		void save(std::string filePath);
		std::string serializeJSON();
		Song& addSong(std::string name, int tempo);
		Song& addSong(std::string name);
		Song& getActiveSong();
		void setSong(std::string name);
		void setGroup(std::string name);
		void setNextGroup(std::string name);
		void serve(int port);
		void setTempo(int bpm) {
			sequencer.setTempo(bpm);
		}
		void setInstrumentPlay(std::string group2, int instrumentId);
		void removeInstrument(int instrumentId);
	private:
		tsal::Mixer masterMixer;
		Sequencer sequencer;
		std::vector<Song> songs;
	};

}

#endif
#endif
#ifndef NOTE_HPP
#define NOTE_HPP

#include <cereal/access.hpp>
#include <cereal/archives/xml.hpp>
#include <cereal/archives/json.hpp>

namespace lotide {


	class Note {
```

```
public:
	Note();
	Note(double n, double v, int s, int d);
	Note(const Note& otherNote);
	double getNote() const;
	double getVelocity() const;
	int getStartTime() const;
	int getDuration() const;
	unsigned getId() const { return id; }
	void setNote(double n);
	void setVelocity(double v);
	void setStartTime(int t);
	void setDuration(int d);
	static unsigned nextId;
	bool operator ==(const Note& rhs) const {
		return id == rhs.id;
	}
	bool operator <(const Note& rhs) const
	{
		return id < rhs.getId();
	}
	bool operator =(const Note& other) const {
		return id == other.getId();
	}
private:
	friend class cereal::access;
	template<class Archive>
	void serialize(Archive& ar)
	{
		ar(cereal::make_nvp("note", note),
		  cereal::make_nvp("velocity", velocity),
		  cereal::make_nvp("startTime", startTime),
		  cereal::make_nvp("duration", duration),
		  cereal::make_nvp("id", id));
	}

	double note;
	double velocity;
	int startTime;
	int duration;
	unsigned id;

};
```

```
}

#endif
#ifndef SERVER_HPP
#define SERVER_HPP

#include <string>
#include <cstring>
#include <iostream>

#include "LoTide.hpp"
#include <nlohmann/json.hpp>

#ifdef _WIN32
	// #include <winsock2.h>
	// #include <Ws2tcpip.h>
	#undef UNICODE

	#define WIN32_LEAN_AND_MEAN

	#include <windows.h>
	#include <winsock2.h>
	#include <ws2tcpip.h>
	#include <stdlib.h>
	#include <stdio.h>

	// Need to link with Ws2_32.lib
	#pragma comment (lib, "Ws2_32.lib")
	// #pragma comment (lib, "Mswsock.lib")

	#define DEFAULT_BUFLEN 512
	#define DEFAULT_PORT "27015"
#else
	#include <sys/socket.h>
	#include <netinet/in.h>
	#include <stdlib.h>
	#include <unistd.h>
#endif

#define MAX_BUF_LENGTH 4096;

namespace lotide {
	class Server {
```

```
        public:
                Server(int port);
                ~Server();
                void init();
                void parseExecute(std::string command, std::vector<std::string> params);
                void initializeLoTide();
        private:
                int mPort;
                unsigned int new_socket;
                unsigned int valread;
                unsigned int server_fd;
                unsigned client_socket;
                unsigned listen_socket;
                struct sockaddr_in address;
                int opt = 1;
                int addrlen = sizeof(address);
                nlohmann::json j;
                #ifdef _WIN32
                        WSADATA wsaData;
                        int iResult;
                #endif
                LoTide lt;
                std::string songState;
                unsigned activeSynthId = 0;
                std::string activeInstanceGroup;
                unsigned phraseIncrement = 0;
                unsigned activePhraseId = 0;
        };
}

#endif
#ifndef SEQUENCER_HPP
#define SEQUENCER_HPP

#include <string>
#include <iostream>
#include <unordered_map>
#include <vector>
#include <chrono>
#include <thread>
#include <list>
#include "tsal.hpp"
#include "Song.hpp"
```

```
namespace lotide {

   class Sequencer {

   public:
      const static int ppq = 8;

      Sequencer(unsigned bpm);
      void addUpcoming(Note note, unsigned synthId);
      void clearAll();
      void start();
      void start(unsigned time);
      void stop();
      void findUpcomingNotes();
      void processNotes();
      void tick();
      void setSong(Song& s);
      Song& getSong() { return *activeSong; }
      void setTempo(int bpm) { mBpm = bpm; }
      void setCurrentTime(int pulse);
   private:
      int mBpm = 90;
      int currentTime;
      Song* activeSong;
      bool isPlaying;
      std::unordered_map<unsigned, std::list<Note>> playing;
      std::map<Note, int> noteTimes;
   };

}

#endif
#ifndef PHRASE_HPP
#define PHRASE_HPP

#include <cereal/access.hpp>
#include <cereal/types/vector.hpp>
#include <cereal/types/string.hpp>
#include <cereal/archives/xml.hpp>
#include <cereal/archives/json.hpp>

#include "Note.hpp"
```

```
#include "tsal.hpp"

namespace lotide {

	class Phrase {

	public:
		Phrase();
		Phrase(std::string name, unsigned id);
		Phrase(std::string newName, unsigned id, Phrase& otherPhrase);
		void setLength(int newLength);
		const std::vector<Note>& getNotes() { return mNotes; }
		unsigned getLength() { return mLength; }
		void addNote(Note&& n);
		void removeNote(double myNote, int startTime);
		unsigned getId() { return mId; }
		std::string getName() { return mName; }
	private:
		friend class cereal::access;
		template<class Archive>
		void serialize(Archive & ar)
		{
			ar(cereal::make_nvp("name", mName),
			  cereal::make_nvp("length", mLength),
			  cereal::make_nvp("id", mId),
			  cereal::make_nvp("notes", mNotes));
		}
		std::string mName;
		unsigned mLength = -1;
		unsigned mId;
		std::vector<Note> mNotes;
	};

}

#endif
```

LoTide - A dynamic music generation framework
==========

LoTide is a modern, open source, music generation framework that can run anywhere. LoTide provides developers an easy way to create seamless-dynamic music.

Start using LoTide

```console
mkdir build
cd build
cmake ..
make
```

```
const net = require('net');
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

rl.setPrompt('lotide> ');

function socketAdd(myStr) {
  const client = net.createConnection({ port: 8198 }, () => {
          console.log('connected to server!');
          client.write(myStr);
  });

  client.on('data', (data) => {
          console.log(data.toString());
          client.end();
  });

  client.on('end', () => {
          rl.prompt();
  });
}

function reversed(r){
  function rev(r, acc){
          if(r=="")
            return acc;
          else
            return rev(r.substr(1), r[0]+acc);
  }
  return rev(r, "");
```

```
}

function removeCommand(str){
  function rc(str, cmd){
          if(str == "" || str.substr(0,1) == " ")
            return cmd;
          else
            return rc(str.substr(1), str.substr(0,1)+cmd);
  }
  return reversed(rc(str, ""));
}

rl.on('line', function(line) {
  var cmd = removeCommand(line);
  switch(cmd) {
  case "play":
                  socketAdd(JSON.stringify({ command: "play", parameters [""] }));
          done();
          break;
  case "stop":
                  socketAdd(JSON.stringify({ command: "stop", parameters [""] }));
          done();
          break;
  case "close":
                  socketAdd(JSON.stringify({ command: "close", parameters [""] }));

          done();
          break;
  default:
          done();
          break;
  }
});

function done() {
  rl.prompt();
}

console.log("Started LoTide CLI.\nEnter LoTide Commands.\nType \"help\" for a list of
commands.\n");

done();
const net = require('net');
```

```
const WebSocket = require('ws');

const wss = new WebSocket.Server({ port: 8080 });

function socketAdd(myStr, ws) {
  const client = net.createConnection({ port: 8198 }, () => {
        console.log('connected to server!');
        client.write(myStr);
  });

  client.on('data', (data) => {
        console.log("From Server");
        console.log(data.toString());
        ws.send(data.toString());
        // client.end();
  });

  client.on('end', () => {
        console.log("Disconnected from Socket.");
  });
}

wss.on('connection', ws => {
  ws.on('message', message => {
        let jsonMessage = JSON.parse(message);
        // if (jsonMessage.command == "play") {
        //   socketAdd(message, ws);
        // } else if (jsonMessage.command == "stop") {
        //   socketAdd(message, ws);
        // }
        if (jsonMessage.command == "close") {
          socketAdd(message, ws);
          wss.close();
        } else {
          console.log(message);
          socketAdd(message, ws);
        }
  });
});

wss.on('close', function() {
  console.log("Websocket Closed");
});
```

```
<!DOCTYPE HTML>
<head>
  <title>LoTide Demonstration</title>
  <!-- <link rel="stylesheet" type="text/css" href="project4.css" /> -->
</head>
<body>
  <div id="mainWindow">
    <canvas id="canvas1" width="1024" height="768" style="background-color: #c1c1c1;
margin: auto; display: block; border: 2px solid black">
      <p>Your browser doesn't support the HTML5 canvas element.</p>
    </canvas>
  </div>
  <script src="http://code.jquery.com/jquery.min.js"></script>
  <script type="text/javascript" src="demo.js"></script>
</body>
</html>
"use strict";

const url = 'ws://localhost:8080';
let connection = new WebSocket(url);
let loaded = false;
var stateObj;

connection.onopen = () => {
        connection.send(JSON.stringify({ command: "load", parameters: ["Song1",
"temp.lot"] }));
};

connection.onerror = (error) => {
  console.log(`WebSocket error: ${error}`);
};

connection.onmessage = (e) => {
        //stateObj = JSON.parse(e.data);
        if (!loaded) {
                loaded = true;
                connection.send(JSON.stringify({ command: "setActiveGroup",
parameters: ["part1"] }));

                connection.send(JSON.stringify({ command: "play", parameters: [""] }));
                drawables[0].stopInstruments();
        }
};
```

```
let first = true;

let speed = 3;

let canvas;
let context;

let drawables = [];
let player;

let keyLeft = false;
let keyRight = false;
let keyDown = false;
let keyUp = false;

class Drawable {
	constructor(x, y) {
		this.x = x;
		this.y = y;
		this.radius = 15;
	}

	draw () {
		context.lineWidth = 3;
		context.fillStyle = (this.hasOwnProperty("color") ? this.color : "#000000");
		context.strokeStyle = "#000000";
		let r = this.radius;
		context.beginPath();
		context.arc(this.x, this.y, r, 0, Math.PI * 2, false);
		context.stroke();
		context.fill();
	}
}

class Emitter extends Drawable {
	constructor(x, y, synthId, hitbox) {
		super(x, y);
		this.synthId = synthId;
		this.radius = 25;
		this.hitbox = hitbox;
	}
```

```
	draw(groupName) {
		super.draw();

		context.globalAlpha = 0.1;
		let res = this.hitbox.draw();
		context.globalAlpha = 1.0;

		if (res[0]) {			//enter
			connection.send(JSON.stringify({ command: "setInstrumentPlay",
parameters: [groupName, ""+this.synthId] }));
		} else if (res[1]) {	//leave
			connection.send(JSON.stringify({ command: "removeInstrument",
parameters: [""+this.synthId] }));
		}
	}

	stopInstruments() {
		connection.send(JSON.stringify({ command: "removeInstrument",
parameters: [""+this.synthId] }));
	}
}

class Rectangle extends Drawable {
	constructor(x, y, bx, by) {
		super(x, y);
		this.bx = bx;
		this.by = by;
		this.playerInside = false;
	}

	draw () {
		this.oldPlayerInside = this.playerInside;

		this.playerInside = this.pointInside(player.x, player.y);

		context.fillStyle = (this.hasOwnProperty("color") ? this.color : "#000000");
		context.fillRect(this.x, this.y, this.bx, this.by);

		return [this.playerInside && this.oldPlayerInside !== this.playerInside,
		(!this.playerInside) && this.oldPlayerInside !== this.playerInside];
	}

	pointInside(x, y) {
```

```
                return (x >= this.x && x <= this.x + this.bx && y >= this.y && y <= this.y +
this.by);
        }
}

class GroupZone extends Rectangle {
        constructor(x, y, bx, by, groupName, emitters, transition) {
                super(x, y, bx, by);
                this.groupName = groupName;
                this.transition = transition;

                this.emitters = emitters;

                this.emitters.forEach(emitter => emitter.color = this.color);
        }

        stopInstruments() {
                this.emitters.forEach(emitter => emitter.stopInstruments());
        }

        draw() {
                let ret = super.draw();

                if (ret[0] === true) {
                        connection.send(JSON.stringify({ command: "setActiveGroup",
parameters: [this.groupName] }));
                        connection.send(JSON.stringify({ command: "setTempo",
parameters: [""+this.tempo] }));
                        this.stopInstruments();
                }

                this.emitters.forEach(emitter => emitter.draw(this.groupName));
        }
}


$(document).ready(function () { init(); });

function moveLeft() {
        if (player != undefined) {
                if (player.x > 0) {
                        player.x = player.x - speed;
                }
```

```
        }
}

function moveUp() {
        if (player != undefined) {
                if (player.y > 0) {
                        player.y = player.y - speed;
                }
        }
}

function moveRight() {
        if (player != undefined) {
                if (player.x < canvas.width) {
                        player.x = player.x + speed;
                }
        }
}

function moveDown() {
        if (player != undefined) {
                if (player.y < canvas.height) {
                        player.y = player.y + speed;
                }
        }
}

$(document).keydown(function(e) {
        switch(e.which) {
                case 37:        //left
                        keyLeft = true;
                        break;

                case 38:        //up
                        keyUp = true;
                        break;

                case 39:        //right
                        keyRight = true;
                        break;

                case 40:        //down
                        keyDown = true;
```

```
                        break;
        }
});

$(document).keyup(function(e) {
        switch(e.which) {
                case 37:        //left
                        keyLeft = false;
                        break;

                case 38:        //up
                        keyUp = false;
                        break;

                case 39:        //right
                        keyRight = false;
                        break;

                case 40:        //down
                        keyDown = false;
                        break;

                case 13:
                        connection.send(JSON.stringify({ command: "stop", parameters:
[""] }));
                        break;
        }
});

function init() {
        canvas = $('#canvas1')[0];
        context = canvas.getContext('2d');

        player = new Drawable(canvas.width / 2 - 45, canvas.height / 2 - 45);
        player.radius = 4;

        drawables[0] = new GroupZone(0, 0, canvas.width / 2, canvas.height / 2,
                "part1",
                [new Emitter(35, 35, 0, new Rectangle(5, 5, 200, 320)),
                new Emitter(445, 165, 1, new Rectangle(80, 95, 415, 180)),
                new Emitter(320, 60, 2, new Rectangle(120, 35, 250, 135))],
                true
        );
```

```
        drawables[0].color = "#eb81b0";
        drawables[0].playerInside = true;
        drawables[0].tempo = 138;

        drawables[1] = new GroupZone(canvas.width / 2, 0, canvas.width, canvas.height /
2,
                "part2",
                [new Emitter(canvas.width / 2 + 35, 65, 1, new Rectangle(canvas.width / 2 +
5, 35, 400, 180)),
                new Emitter(canvas.width / 2 + 450, 240, 2, new Rectangle(canvas.width / 2
+ 200, 120, 285, 220))],
                true
        );
        drawables[1].color = "#ebebb0";
        drawables[1].playerInside = true;
        drawables[1].tempo = 122;

        drawables[2] = new GroupZone(0, canvas.height / 2, canvas.width, canvas.height /
2,
                "part3",
                [new Emitter(65, canvas.height / 2 + 85, 0, new Rectangle(35, canvas.height
/ 2 + 65, 800, 180)),
                new Emitter(240, canvas.height / 2 + 285, 1, new Rectangle(210,
canvas.height / 2 + 125, 450, 240)),
                new Emitter(430, canvas.height / 2 + 35, 2, new Rectangle(380,
canvas.height / 2 + 10, 250, 275))],
                true
        );
        drawables[2].color = "#b081eb";
        drawables[2].playerInside = true;
        drawables[2].tempo = 110;



        // drawables[1] = new GroupZone(canvas.width / 2, 0, canvas.width, canvas.height
/ 2,
        //      "part1",
        //      [new Emitter(canvas.width / 2 + 100, 15, 0, "#000000"),
        //      new Emitter(canvas.width / 2 + 15, 100, 1, "#000000")],
        //      true
        // );
        // drawables[1].color = "#83eb81";
        // // drawables[2] = new Rectangle(0, canvas.height / 2, 0, canvas.height);
```

```
        // // drawables[2].color = "#eb81e0";
        // drawables[2] = new GroupZone(0, canvas.height / 2, canvas.width,
canvas.height,
        //      "part1",
        //      [new Emitter(100, canvas.height / 2 + 15, 0, "#000000"),
        //      new Emitter(15, canvas.height / 2 + 100, 1, "#000000")],
        //      true
        // );
        // drawables[2].color = "#9381eb";

        setInterval(draw, 15);
}

function draw() {
        if (keyLeft) {moveLeft();}
        if (keyRight) {moveRight();}
        if (keyDown) {moveDown();}
        if (keyUp) {moveUp();}

        context.clearRect(0, 0, canvas.width, canvas.height);

        if (loaded && drawables != undefined && context != undefined) {
                drawables.forEach(drawable => {
                        if (typeof drawable.draw === "function") {drawable.draw();}
                });
        }

        player.draw();
}
const url = 'ws://localhost:8080';
const connection = new WebSocket(url);
let jsonStr;
let stateObj;
let groupName;
let currentLength = 0;
let activeNotes = [];
let activateSel = 0;

connection.onopen = () => {
  // if (groups) {
  connection.send(JSON.stringify({ command: "addSong", parameters: ["default"] }));
  connection.send(JSON.stringify({ command: "addSynth", parameters: [""] }));
  connection.send(JSON.stringify({ command: "addGroup", parameters: ["normal"] }));
```

```
  connection.send(JSON.stringify({ command: "addPhrase", parameters: ["Phrase1"] }));
  connection.send(JSON.stringify({ command: "setLength", parameters: ["384"] }));
  connection.send(JSON.stringify({ command: "setActiveGroup", parameters: ["normal"]
}));
  // }
};



connection.onerror = (error) => {
  console.log(`WebSocket error: ${error}`);
};

connection.onmessage = (e) => {
  stateObj = JSON.parse(e.data);
  console.log(stateObj);
  phrases = stateObj.song.phrases;
  groups = stateObj.song.groups;
  synths = stateObj.song.synths;
  if (activateSel == 0) {
        updateSelectors(phrases, groups, synths);
        activateSel++;
  }
};

if (jsonStr != undefined) {

}

const modulationSelector = document.getElementById("modulation-mode-selector");
const oscillator1Selector = document.getElementById("osc1-mode-selector");
const oscillator2Selector = document.getElementById("osc2-mode-selector");

const noteSizeSelector = document.getElementById("note-size-selector");
const timeSignatureSelector = document.getElementById("time-signature-selector");


const volumeSlider = document.getElementById("volume-slider-range");
const bpmSlider = document.getElementById("bpm-slider-range");
const attackSlider = document.getElementById("attack-slider-range");
const sustainSlider = document.getElementById("sustain-slider-range");
const decaySlider = document.getElementById("decay-slider-range");
const releaseSlider = document.getElementById("release-slider-range");
```

```
const offsetSlider = document.getElementById("offset-slider-range");

const playButton = document.getElementById("play-button");
const stopButton = document.getElementById("stop-button");
const closeButton = document.getElementById("close-button");

const increaseMeasureButton = document.getElementById("increase-measure-button");

const updateButton = document.getElementById("update-button");

// const setGroupButton = document.getElementById("set-group-button");
const phraseButton = document.getElementById("phrase-button");
const synthButton = document.getElementById("synth-button");
const songButton = document.getElementById("song-button");

playButton.onclick = () => {
  connection.send(JSON.stringify({ command: "play", parameters: [""] }));
  // myMove();
};
stopButton.onclick = () => {
  connection.send(JSON.stringify({ command: "stop", parameters: [""] }));
};
closeButton.onclick = () => {
  connection.send(JSON.stringify({ command: "close", parameters: [""] }));
};

increaseMeasureButton.onclick = () => {
  // currentLength += 32;
  // connection.send(JSON.stringify({ command: "setActiveGroup", parameters:
[groups[0].value0] }));
};

updateButton.onclick = () => {
  // currentLength = stateObj.song.current_length;
  connection.send(JSON.stringify({ command: "playNote", parameters: ["60", "100"] }));
  connection.send(JSON.stringify({ command: "setLength", parameters: ["384"] }));
  updateElements(192);
  // connection.send(JSON.stringify({ command: "setActiveSynth", parameters:
[groupName] }));
  // connection.send(JSON.stringify({ command: "setActiveGroup", parameters:
[groups[0].value0] }));
  // console.log("update" + currentLength);
  // if (currentLength != 0)
```

```
};


noteSizeSelector.addEventListener('change', (event) => {
  connection.send(JSON.stringify({ command: "setLength", parameters: ["384"] }));
  updateElements(192);
});

timeSignatureSelector.addEventListener('change', (event) => {
  connection.send(JSON.stringify({ command: "setLength", parameters: ["384"] }));
  updateElements(192);
});

modulationSelector.addEventListener('change', (event) => {
  connection.send(JSON.stringify({ command: "setModulationMode", parameters:
[event.target.value.toString()] }));
});

oscillator1Selector.addEventListener('change', (event) => {
  connection.send(JSON.stringify({ command: "setOSC1Mode", parameters:
[event.target.value.toString()] }));
});

oscillator2Selector.addEventListener('change', (event) => {
  connection.send(JSON.stringify({ command: "setOSC2Mode", parameters:
[event.target.value.toString()] }));
});

modulationSelector.addEventListener('change', (event) => {
  connection.send(JSON.stringify({ command: "setModulationMode", parameters:
[event.target.value.toString()] }));
});

bpmSlider.addEventListener('input', () => {
  connection.send(JSON.stringify({ command: "setLength", parameters:
[bpmSlider.value.toString()]}));
}, false);


volumeSlider.addEventListener('input', () => {
  connection.send(JSON.stringify({ command: "setVolume", parameters:
[volumeSlider.value.toString()]}));
}, false);
```

```
attackSlider.addEventListener('input', () => {
  connection.send(JSON.stringify({ command: "setAttack", parameters:
[attackSlider.value.toString()]}));
}, false);

decaySlider.addEventListener('input', () => {
  connection.send(JSON.stringify({ command: "setDecay", parameters:
[attackSlider.value.toString()]}));
}, false);

releaseSlider.addEventListener('input', () => {
  connection.send(JSON.stringify({ command: "setRelease", parameters:
[attackSlider.value.toString()]}));
}, false);

sustainSlider.addEventListener('input', () => {
  connection.send(JSON.stringify({ command: "setSustain", parameters:
[attackSlider.value.toString()]}));
}, false);

offsetSlider.addEventListener('input', () => {
  connection.send(JSON.stringify({ command: "setOSC2Offset", parameters:
[attackSlider.value.toString()]}));
}, false);

synthButton.onclick = () => {
  connection.send(JSON.stringify({ command: "addSynth", parameters: [""] }));
};

const groupModal = document.getElementById("group-modal");
const groupButton = document.getElementById("group-button");
const span = document.getElementById("close0");
const groupForm = document.getElementById('group-form');

groupButton.onclick = () => {
  groupModal.style.display = "block";
};

span.onclick = () => {
  groupModal.style.display = "none";
};
```

```
window.onclick = (e) => {
  if (event.target == groupModal) {
        groupModal.style.display = "none";
  }
};

groupForm.addEventListener('submit', e => {

  const groupInput = document.getElementById("group-name");

  groupName = groupInput.value;
  connection.send(JSON.stringify({ command: "addGroup", parameters:
[groupInput.value] }));
  connection.send(JSON.stringify({ command: "setActiveGroup", parameters:
[groupInput.value] }));
  e.preventDefault();
  groupModal.style.display = "none";
});

const songModal = document.getElementById("song-modal");
const span1 = document.getElementById("close1");
const songForm = document.getElementById('song-form');

songButton.onclick = () => {
  songModal.style.display = "block";
};

span1.onclick = () => {
  songModal.style.display = "none";
};

window.onclick = (e) => {
  if (event.target == songModal) {
        songModal.style.display = "none";
  }
};

songForm.addEventListener('submit', e => {
  const songInput = document.getElementById('song-name');
  connection.send(JSON.stringify({ command: "addSong", parameters: [songInput.value]
}));
  e.preventDefault();
```

```
  songModal.style.display = "none";
});

const phraseModal = document.getElementById("phrase-modal");
const span2 = document.getElementById("close2");
const phraseForm = document.getElementById('phrase-form');

phraseButton.onclick = () => {
  phraseModal.style.display = "block";
};

span2.onclick = () => {
  phraseModal.style.display = "none";
};

window.onclick = (e) => {
  if (e.target == phraseModal) {
          phraseModal.style.display = "none";
  }
};

phraseForm.addEventListener('submit', e => {
  const phraseInput = document.getElementById('phrase-name');
  connection.send(JSON.stringify({ command: "addPhrase", parameters:
[phraseInput.value] }));
  e.preventDefault();
  phraseModal.style.display = "none";
});

var notes = {C8: 108, B7: 107, As7: 106, A7: 105, Gs7: 104, G7: 103, Fs7: 102, F7: 101,
                        E7: 100, Ds7: 99, D7: 98, Cs7: 97, C7: 96, B6: 95, As6: 94, A6: 93,
Gs6: 92,
                        G6: 91, Fs6: 90, F6: 89, E6: 88, Ds6: 87, D6: 86, Cs6: 85, C6: 84, B5:
83,
                        As5: 82, A5: 81, Gs5: 80, G5: 79, Fs5: 78, F5: 77, E5: 76, Ds5: 75, D5:
74,
                        Cs5: 73, C5: 72, B4: 71, As4: 70, A4: 69, Gs4: 68, G4: 67, Fs4: 66, F4:
65, E4: 64,
                        Ds4: 63, D4: 62, Cs4: 61, C4: 60, B3: 59, As3: 58, A3: 57, Gs3: 56, G3:
55, Fs3: 54,
                        F3: 53, E3: 52, Ds3: 51, D3: 50, Cs3: 49, C3: 48, B2: 47, As2: 46, A2:
45, Gs2: 44, G2: 43,
```

```
                    Fs2: 42, F2: 41, E2: 40, Ds2: 39, D2: 38, Cs2: 37, C2: 36, B1: 35, As1:
34, A1: 33,
                    Gs1: 32, G1: 31, Fs1: 30, F1: 29, E1: 28, Ds1: 27, D1: 26, Cs1: 25, C1:
24, B0: 23, As0: 22, A0: 21};

function updateElements(fullLength) {
  let myNotes = document.getElementById("my-notes");
  myNotes.innerHTML = '';

  let volume = document.getElementById("volume-slider-range").value;
  let velocity = document.getElementById("velocity-slider-range").value;
  let noteSize = document.getElementById("note-size-selector-select").value;
  let timeSig = document.getElementById("time-signature-selector-select").value;

  if (myNotes.innerHTML == '') {
        for (let i in notes) {
          let noteRow = document.createElement("div");
          noteRow.style.border = "0";
          noteRow.style.display = "flex";
          noteRow.style.flexDirection = "row";

          noteRow.style.color= "#fff";
          noteRow.onmouseover = () => {
                noteRow.style.color= "#323652";
          };

          noteRow.onmouseleave = () => {
                noteRow.style.color= "#e7e7e7";
          };
          myNotes.append(noteRow);
          let noteRowNote = document.createElement("div");
          noteRowNote.innerHTML = i;
          noteRowNote.style.marginTop = "200px";
          noteRowNote.style.border = "0px";
          noteRowNote.style.width = "30px";
          noteRowNote.style.height = "20px";
          noteRowNote.style.padding = "0px";
          noteRowNote.style.margin = "0px";


          noteRowNote.style.backgroundColor = "#191B28";
          // noteRowNote.style.border = "thin solid #6A6F8D";
          noteRow.append(noteRowNote);
```

```
        let singleNote = 0;
        for (let j = 0; j < fullLength/8; ++j) {
                let noteSections = document.createElement("div");
                noteSections.style.width = "64px";
                noteSections.style.height = "20px";
                noteSections.id = i + "-" + j;

                noteSections.style.display = "flex";
                noteSections.style.flexDirection = "row";
                if ((j + 1) % timeSig == 0) {
                  // noteSections.style.border = "thin solid #6A6F8D";
                  noteSections.style.borderRight = "thick solid #6A6F8D";
                } else {
                  noteSections.style.borderRight = "thin solid #6A6F8D";
                }

                if (activeNotes.includes(i + "-" + j)) {
                  noteSections.style.background = "rgba(41, 45, 63, 1)";
                } else {
                  noteSections.style.background = "rgba(66, 72, 98, 0.9)";
                }
                singleNote += 8;

                noteRow.append(noteSections);

                if (noteSize == 16) {
                  noteSections.addEventListener('click', (evt) => {
                        if (noteSections.style.backgroundColor != "rgba(66, 72, 98, 0.9)") {
                          noteSections.style.background = "rgba(66, 72, 98, 0.9)";
                          connection.send(JSON.stringify({ command: "removeNote",
parameters: [notes[i].toString(), j * noteSize + ""] }));
                          activeNotes = activeNotes.filter(e => {
                                return e != (i + "-" + j);
                          });
                        } else {
                          noteSections.style.background = "rgba(41, 45, 63, 1)";
                          connection.send(JSON.stringify({ command: "addNote",
parameters: [notes[i].toString(), velocity, j * 16 + "", noteSize] }));
                          connection.send(JSON.stringify({ command: "setActiveGroup",
parameters: [groups[0].value0] }));
                          connection.send(JSON.stringify({ command: "setLength",
parameters: ["384"] }));
                          activeNotes.push(i + "-" + j);
```

```
                        }
                        connection.send(JSON.stringify({ command: "playNote",
parameters: [notes[i].toString(), velocity] }));
                    });
                  }


                  for (let k = 0; k < 2; ++k) {
                    let note8 = document.createElement("div");
                    note8.style.display = "flex";
                    note8.style.flexDirection = "row";
                    note8.style.width = "32px";
                    note8.style.height = "20px";
                    note8.id = i + "-" + j + "-" + k;
                    noteSections.append(note8);

                    if (activeNotes.includes(i + "-" + j + "-" + k)) {
                          note8.style.background = "rgba(41, 45, 63, 1)";
                    } else {
                          note8.style.background = "rgba(66, 72, 98, 0)";
                    }
                    if (noteSize == 8) {
                          note8.addEventListener('click', (evt) => {
                            // if (note8.style.backgroundColor != "rgb(66, 72, 98)" &&
noteSections.style.backgroundColor != "rgb(66, 72, 98)") {
                            if (note8.style.backgroundColor != "rgba(66, 72, 98, 0)") {
                                  note8.style.background = "rgba(66, 72, 98, 0)";
                                  connection.send(JSON.stringify({ command: "removeNote",
parameters: [notes[i].toString(), j *  16 + k * 8 + ""] }));
                                  activeNotes = activeNotes.filter(e => {
                                    return e != (i + "-" + j + "-" + k);
                                  });
                            } else {
                                  // if (noteSections.style.backgroundColor == "rgb(66, 72, 98)")
{
                                  //   noteSections.style.background = "#292D3F";
                                  //   connection.send(JSON.stringify({ command:
"removeNote", parameters: [notes[i].toString(), j * noteSize + ""] }));
                                  // }

                                  note8.style.background = "rgba(41, 45, 63, 1)";
                                  connection.send(JSON.stringify({ command: "addNote",
parameters: [notes[i].toString(), velocity, j * 16 + k * 8 + "", noteSize] }));
```

```
                                connection.send(JSON.stringify({ command:
"setActiveGroup", parameters: [groups[0].value0] }));
                                connection.send(JSON.stringify({ command: "setLength",
parameters: ["384"] }));
                                activeNotes.push(i + "-" + j + "-" + k);
                            }
                            connection.send(JSON.stringify({ command: "playNote",
parameters: [notes[i].toString(), velocity] }));
                        });
                }

                for (let m = 0; m < 2; ++m) {
                        let note16 = document.createElement("div");
                        note16.style.display = "flex";
                        note16.style.flexDirection = "row";
                        note16.style.width = "16px";
                        note16.style.height = "20px";
                        note8.append(note16);
                        note16.id = i + "-" + j + "-" + k + "-" + m;

                        if (activeNotes.includes(i + "-" + j + "-" + k + "-" + m )) {
                          note16.style.background = "rgba(41, 45, 63, 1)";
                        } else {
                          note16.style.background = "rgba(66, 72, 98, 0)";
                        }

                        if (noteSize == 4) {
                          note16.addEventListener('click', (evt) => {
                                // if (note16.style.backgroundColor != "rgb(66, 72, 98)" &&
noteSections.style.backgroundColor != "rgb(66, 72, 98)" && note8.style.backgroundColor
!= "rgb(66, 72, 98)") {
                                if (note16.style.backgroundColor != "rgba(66, 72, 98, 0)") {
                                  note16.style.background = "rgba(66, 72, 98, 0)";
                                  connection.send(JSON.stringify({ command: "removeNote",
parameters: [notes[i].toString(), j * 16 + k * 8 + m * 4 + ""] }));
                                  activeNotes = activeNotes.filter(e => {
                                        return e != (i + "-" + j + "-" + k + "-" + m);
                                  });
                                } else {
                                  // if (noteSections.style.backgroundColor == "rgb(66, 72, 98)"
|| note8.style.backgroundColor == "rgb(66, 72, 98)") {
                                  //      noteSections.style.background = "#292D3F";
                                  //      note8.style.background = "#292D3F";
```

```
                                    //      note8.style.opacity = "0";
                                    //      connection.send(JSON.stringify({ command:
"removeNote", parameters: [notes[i].toString(), j * noteSize + ""] }));
                                    // }
                                    note16.style.background = "rgba(41, 45, 63, 1)";
                                    connection.send(JSON.stringify({ command: "addNote",
parameters: [notes[i].toString(), velocity, j * 16 + k * 8 + m * 4 + "", noteSize] }));
                                    connection.send(JSON.stringify({ command:
"setActiveGroup", parameters: [groups[0].value0] }));
                                    connection.send(JSON.stringify({ command: "setLength",
parameters: ["384"] }));
                                    activeNotes.push(i + "-" + j + "-" + k + "-" + m );
                                    }
                                  connection.send(JSON.stringify({ command: "playNote",
parameters: [notes[i].toString(), velocity] }));
                              });
                            }

                            for (let n = 0; n < 2; ++n) {
                              let note32 = document.createElement("div");
                              note32.style.display = "flex";
                              note32.style.flexDirection = "row";
                              note32.style.width = "8px";
                              note32.style.height = "20px";
                              note32.id = i + "-" + j + "-" + k + "-" + m + "-" + n;
                              note16.append(note32);

                              if (activeNotes.includes(i + "-" + j + "-" + k + "-" + m + "-" + n )) {
                                    note32.style.background = "rgba(41, 45, 63, 1)";
                              } else {
                                    note32.style.background = "rgba(66, 72, 98, 0)";
                              }

                              if (noteSize == 2) {
                                    note32.addEventListener('click', (evt) => {
                                      if (note32.style.backgroundColor != "rgba(66, 72, 98, 0)") {
                                      // if (note32.style.backgroundColor != "rgb(66, 72, 98)" &&
noteSections.style.backgroundColor != "rgb(66, 72, 98)" &&
note16.style.backgroundColor != "rgb(66, 72, 98)" && note8.style.backgroundColor !=
"rgb(66, 72, 98)") {
                                            note32.style.background = "rgba(66, 72, 98, 0)";
```

```
                                        connection.send(JSON.stringify({ command:
"addNote", parameters: [notes[i].toString(), velocity, j * 16 + k * 8 + m * 4 + n * 2 + "",
noteSize] }));
                                        connection.send(JSON.stringify({ command:
"setActiveGroup", parameters: [groups[0].value0] }));
                                        connection.send(JSON.stringify({ command:
"setLength", parameters: ["384"] }));
                                        activeNotes = activeNotes.filter(e => {
                                          return e != (i + "-" + j + "-" + k + "-" + m + "-" + n);
                                        });
                                    } else {
                                        // if (noteSections.style.backgroundColor == "rgb(66,
72, 98)" || note8.style.backgroundColor == "rgb(66, 72, 98)" ||
note16.style.backgroundColor == "rgb(66, 72, 98)") {
                                        //   noteSections.style.background = "#292D3F";
                                        //   note8.style.background = "#292D3F";
                                        //   note16.style.background = "#292D3F";
                                        //   connection.send(JSON.stringify({ command:
"removeNote", parameters: [notes[i].toString(), j * noteSize + ""] }));
                                        // }
                                        note32.style.background = "rgba(41, 45, 63, 1)";
                                        connection.send(JSON.stringify({ command:
"removeNote", parameters: [notes[i].toString(), j * 16 + k * 8 + m * 4 + n * 2 + ""] }));
                                        console.log(j * noteSize);
                                        console.log(activeNotes);
                                        activeNotes.includes(i + "-" + j + "-" + k + "-" + m + "-"
+ n );
                                    }
                                    connection.send(JSON.stringify({ command: "playNote",
parameters: [notes[i].toString(), velocity] }));
                                });
                            }
                        }
                    }
                }
            }
        }
    }
}

function updateSelectors(phrases, groups, synths) {
  let phraseSelector = document.getElementById("phrase-selector-select");
  let groupSelector = document.getElementById("group-selector-select");
```

```
  let synthSelector = document.getElementById("synth-selector-select");
  phraseSelector.innerHTML = "";
  groupSelector.innerHTML = "";
  synthSelector.innerHTML = "";

  if (phraseSelector.innerHTML == "") {
          for (phrase of phrases) {
            let phraseOption = document.createElement("option");
            phraseOption.innerHTML = phrase.name;
            phraseOption.value = phrase.name;
            phraseSelector.append(phraseOption);
          }
  }

  phraseSelector.addEventListener('change', (event) => {
          connection.send(JSON.stringify({ command: "setActivePhrase", parameters:
[phrases.findIndex(e =>  e.name == event.target.value).toString()] }));
  });

  if (groupSelector.innerHTML == "") {
          for (group of groups) {
            let groupOption = document.createElement("option");
            groupOption.innerHTML = group.value0;
            groupOption.value = group.value0;
            groupSelector.append(groupOption);
          }
  }

  groupSelector.addEventListener('change', (event) => {
          connection.send(JSON.stringify({ command: "setActiveGroup", parameters:
[event.target.value] }));
  });

  if (synthSelector.innerHTML == "") {
          for (let i = 0; i < synths.length; ++i) {
            let synthOption = document.createElement("option");
            synthOption.innerHTML = i;
            synthOption.value = i;
            synthSelector.append(synthOption);
          }
  }

  synthSelector.addEventListener('change', (event) => {
```

```
        connection.send(JSON.stringify({ command: "setActiveSynth", parameters:
[event.target.value.toString()] }));
  });
}

// const playRow = document.getElementById("play-row");

// function playing () {
//   var counter = 0;
//   var i = setInterval(function(){
//         counter++;
//         if(counter === 10) {
//           clearInterval(i);
//         }
//   }, 200);
// }
// let pos = 30;
// const elem = document.getElementById("animate");
// let id = setInterval(frame, 5);

// function myMove() {
//   function frame() {
//         if (currentLength) {
//           if (pos == (8 * 64)) {
//                 console.log(pos);
//                 clearInterval(id);
//           } else {
//                 pos++;
//                 console.log(pos);
//                 elem.style.left = pos + 'px';
//           }
//         }
//   }

// var elem = document.getElementById('animate');
// elem.onclick = toggleAnimation;
// elem.style.webkitAnimationPlayState = 'running';

// function toggleAnimation() {
//   var style;
//   style = elem.style;
//   if (style.webkitAnimationPlayState === 'running') {
//         style.webkitAnimationPlayState = 'paused';
```

```
//         document.body.className = 'paused';
//   } else {
//         style.webkitAnimationPlayState = 'running';
//         document.body.className = '';
//   }
// }
//   function myStop() {
//   }
// }
<!doctype html>
<html lang="en">
  <head>
        <meta charset="UTF-8"/>
        <title>Document</title>
        <link href="style.css" rel="stylesheet"/>
        <link
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.13.0/css/all.min.css"
rel="stylesheet">

  </head>
  <body>
        <div id="lotide">LoTide</div>
        <div class="panel">
          <div class="subpanel-upper">
                <button id="update-button"><i class="fas fa-sync"></i></button>
                <button id="play-button"><i class="fas fa-play"></i></button>
                <button id="stop-button"><i class="fas fa-pause"></i></button>
                <button id="close-button"><i class="fas fa-unlink"></i></button>

                <button id="increase-measure-button">Increase Measure</button>

                <button id="group-button">Add Group</button>

                <div id="group-modal" class="modal">
                  <div class="modal-content">
                        <span id="close0" class="close">&times;</span>
                        <form id="group-form">
                          <label>Group Name <input id="group-name" type="text"></label>
                          <br><br>
                          <button type="submit">Add</button>
                        </form>
                  </div>
                </div>
```

```
        <button id="phrase-button">Add Phrase</button>

        <div id="phrase-modal" class="modal">
          <div class="modal-content">
                <span id="close2" class="close">&times;</span>
                <form id="phrase-form">
                  <label>Phrase Name <input id="phrase-name" type="text"></label>
                  <br><br>
                  <button type="submit">Add</button>
                </form>
          </div>
        </div>

        <button id="synth-button">Add Synth</button>

        <button id="song-button">Add Song</button>

        <div id="song-modal" class="modal">
          <div class="modal-content">
                <span id="close1" class="close">&times;</span>
                <form id="song-form">
                  <label>Song Name <input id="song-name" type="text"></label>
                  <br><br>
                  <button type="submit">Add</button>
                </form>
          </div>
        </div>
</div>

<div class="subpanel-upper">
        <div id="selectors">
          <div class="selector-container">
                Groups
                <div id="group-selector">
                  <select id="group-selector-select">
                  </select>
                </div>
          </div>

          <div class="selector-container">
                Phrases
                <div id="phrase-selector">
```

```
          <select id="phrase-selector-select">
          </select>
        </div>
</div>

<div class="selector-container">
        Synths
        <div id="synth-selector">
          <select id="synth-selector-select">
          </select>
        </div>
</div>

<div class="selector-container">
        Note Sizes
        <div id="note-size-selector">
          <select id="note-size-selector-select">
                <option value="16">4</option>
                <option value="8">8</option>
                <option value="4">16</option>
                <option value="2">32</option>
          </select>
        </div>
</div>

<div class="selector-container">
        Time Signatures
        <div id="time-signature-selector">
          <select id="time-signature-selector-select">
                <option value="4">4</option>
                <option value="3">3</option>
          </select>
        </div>
</div>

<div class="selector-container">
        Modulation Mode
        <div id="modulation-mode-selector">
          <select id="modulation-selector-select">
                <option value="0">None</option>
                <option value="1">Mix</option>
                <option value="2">AM</option>
                <option value="3">PM</option>
```

```
            </select>
          </div>
        </div>

        <div class="selector-container">
          Oscillator 1 Mode
          <div id="osc1-mode-selector">
            <select id="osc1-selector-select">
              <option value="0">Sine</option>
              <option value="1">Saw</option>
              <option value="2">Square</option>
              <option value="3">White Noise</option>
            </select>
          </div>
        </div>

        <div class="selector-container">
          Oscillator 2 Mode
          <div id="osc2-mode-selector">
            <select id="osc2-selector-select">
              <option value="0">Sine</option>
              <option value="1">Saw</option>
              <option value="2">Square</option>
              <option value="3">White Noise</option>
            </select>
          </div>
        </div>

        <div class="selector-container">
          Note Size
          <div id="note-size-selector">
            <select id="note-size-selector-select">
              <option value="0">4</option>
              <option value="1">8</option>
              <option value="2">16</option>
              <option value="3">32</option>
            </select>
          </div>
        </div>
      </div>
    </div>
    <div class="subpanel">
      <div id="sliders">
```

```
                <div class="slider-container">
                        Volume
                        <div id="volume-slider">
                          <input type="range" min="0" max="100" value="50" class="slider"
id="volume-slider-range">
                        </div>

                </div>
                <div class="slider-container">
                        Velocity
                        <div id="velocity-slider">
                          <input type="range" min="1" max="127" value="100"
class="slider" id="velocity-slider-range">
                        </div>
                </div>

                <div class="slider-container">
                        BPM
                        <div id="bpm-slider">
                          <input type="range" min="1" max="300" value="90" class="slider"
id="bpm-slider-range">
                        </div>
                </div>

              </div>
        </div>
        <div class="subpanel">
                <div id="sliders">
                 <div class="slider-container">
                        Attack
                        <div id="attack-slider">
                          <input type="range" min="0" max="1" value="0.3" step="0.1"
class="slider" id="attack-slider-range">
                        </div>
                </div>

                <div class="slider-container">
                        Sustain
                        <div id="sustain-slider">
                          <input type="range" min="0" max="1" value="0.3" step="0.1"
class="slider" id="sustain-slider-range">
                        </div>
                </div>
```

```
                <div class="slider-container">
                    Decay
                    <div id="decay-slider">
                      <input type="range" min="0" max="1" value="0.3" step="0.1"
class="slider" id="decay-slider-range">
                    </div>
                </div>

                <div class="slider-container">
                    Release
                    <div id="release-slider">
                      <input type="range" min="0" max="1" value="0.3" step="0.1"
class="slider" id="release-slider-range">
                    </div>
                </div>

                <div class="slider-container">
                    Offset
                    <div id="offset-slider">
                      <input type="range" min="0" max="1" value="0.3" step="0.1"
class="slider" id="offset-slider-range">
                    </div>
                </div>
              </div>
          </div>

          <!-- <div id="play-row"> -->
          <!--    <div id="animate"> -->
          <!--    </div> -->
          <!-- </div> -->
        </div>

        <div id="my-notes" style="display:flex; flex-direction: column"></div>
        <script src="main.js"></script>
  </body>
</html>
@import url(https://fonts.googleapis.com/css?family=Roboto+Condensed);

body, html {
  font-family: 'Roboto', sans-serif;
  overflow: scroll;
  margin: 0px;
```

```
  padding: 0px;
  background-color: #545971;
}

#lotide {
  position: fixed;
  left: 15px;
  top: 15px;
  color: #fff;
  z-index: 100000;
  font-size: 40px;
}

.panel {
  background-color: #222432;
  position: fixed;
  z-index: 100;
  width: 100%;
  height: 250px;
  display: flex;
  flex-direction: column;
  color: #fff;
}

.selector-container {
  padding: 5px;
  text-align: center;
}

.subpanel-upper {
  padding: 10px;
  height: 40px;
  width: 100%;
  display: flex;
  flex-direction: row;
  justify-content: center;
}

.slider-container {
  text-align: center;
  padding: 10px;
}
```

```
.slider {
  -webkit-appearance: none;
  appearance: none;
  width: 100%;
  height: 25px;
  background: #191B28;

  outline: none;
  opacity: 0.7;
  -webkit-transition: .2s;
  transition: opacity .2s;
}

button {
  background-color: #191B28;
  border: none;
  color: #aaa;
  padding: 15px 22px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 12px;
}


button:hover {
  color: white;
}

.slider:hover {
  opacity: 1;
}

.slider::-webkit-slider-thumb {
  -webkit-appearance: none;
  appearance: none;
  width: 25px;
  height: 25px;
  background: #3A3E60;
  cursor: pointer;
}

.slider::-moz-range-thumb {
```

```
  width: 25px; /* Set a specific slider handle width */
  height: 25px; /* Slider handle height */
  background: #4CAF50; /* Green background */
  cursor: pointer; /* Cursor on hover */
}

.subpanel {
  width: 100%;
  height: 60px;
  /* background-color: #222432; */
  display: flex;
  flex-direction: row;
  justify-content: center;
}

#play-row {
  width: 100%;
  height: 20px;
  background-color: #222432;
  position: relative;
  display: flex;
  flex-direction: row;
}

#sliders {
  display: flex;
  flex-direction: row;
}

#selectors {
  display: flex;
  flex-direction: row;
}

.modal {
  display: none; /* Hidden by default */
  position: fixed; /* Stay in place */
  z-index: 1; /* Sit on top */
  padding-top: 300px; /* Location of the box */
  left: 0;
  top: 0;
  width: 100%; /* Full width */
  height: 100%; /* Full height */
```

```
  overflow: auto; /* Enable scroll if needed */
  background-color: rgb(0,0,0); /* Fallback color */
  background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
}


/* Modal Content */
.modal-content {
  background-color: #fefefe;
  margin: auto;
  padding: 20px;
  border: 1px solid #888;
  width: 15%;
}

#my-notes {
  position: absolute;
  margin-top: 250px;
  background-color: #545971;
}

/* The Close Button */
.close {
  color: #aaaaaa;
  float: right;
  font-size: 28px;
  font-weight: bold;
}

.close:hover,
.close:focus {
  color: #000;
  text-decoration: none;
  cursor: pointer;
}

#animate {
  width: 10px;
  height: 20px;
  left: 30px;
  position: absolute;
  background: red;
}
```

```
{
  "name": "lotide-node",
  "version": "1.0.0",
  "description": "Node.js client for LoTide.",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "ws": "^7.2.3"
  }
}
# lotide-node-client
lotide-node-client
```bash
npm install
node server.js
```
#include "AudioBuffer.hpp"
#include "tsal.hpp"
#include <cassert>

using namespace tsal;

void testConstructors() {
  unsigned channels = 2;
  unsigned frames = 1000;
  AudioBuffer<int> a1;
  a1.setChannelCount(channels);
  a1.setFrameCount(frames);

  AudioBuffer<int> a2(frames, channels);
  assert(a1.getChannelCount() == a2.getChannelCount());
  assert(a1.getFrameCount() == a1.getFrameCount());
  assert(a1.size() == a2.size());
  assert(a1.size() == channels * frames);
}

void testResizing() {
  unsigned channels = 2;
```

```
  unsigned frames = 1000;

  AudioBuffer<int> a1(frames, channels);
  AudioBuffer<int> a2;
  a2.setSize(a1);
  assert(a1.size() == a2.size());

  AudioBuffer<int> a3(2000, 4);
  a3.setSize(frames, channels);

  assert(a1.size() == a3.size());
}

void testInterleavingStereo() {
  std::vector<AudioBuffer<int>> buffers(2, AudioBuffer<int>(2, 1));
  assert(buffers.size() == 2);
  for (auto& buffer : buffers) {
    assert(buffer.getChannelCount() == 1);
    assert(buffer.getFrameCount() == 2);
    assert(buffer.size() == 2);
  }

  AudioBuffer<int> a1(2, 2);
  for (unsigned i = 0; i < a1.size(); i++) {
    a1[i] = i;
  }

  std::vector<AudioBuffer<int>*> buffer_pointers;
  for (auto& buffer : buffers) {
    buffer_pointers.push_back(&buffer);
  }
  a1.deinterleaveBuffer(buffer_pointers);

  for (unsigned j = 0; j < buffers.size(); j++) {
    auto& buffer = buffers[j];
    for (unsigned i = 0; i < buffer.size(); i++) {
      assert(buffer[i] == (int) (j + i * 2));
    }
  }

  a1.interleaveBuffers(buffer_pointers);
  for (unsigned i = 0; i < a1.size(); i++) {
    assert(a1[i] == (int) i);
```

```
  }
}

void testInterleavingMono() {
  std::vector<AudioBuffer<int>> buffers(1, AudioBuffer<int>(2, 1));
  assert(buffers.size() == 1);
  for (auto& buffer : buffers) {
    assert(buffer.getChannelCount() == 1);
    assert(buffer.getFrameCount() == 2);
    assert(buffer.size() == 2);
  }

  AudioBuffer<int> a1(2, 2);
  for (unsigned i = 0; i < a1.size(); i++) {
    a1[i] = i * 2;
    std::cout << "a: " << a1[i] << std::endl;
  }

  std::vector<AudioBuffer<int>*> buffer_pointers;
  for (auto& buffer : buffers) {
    buffer_pointers.push_back(&buffer);
  }
  a1.deinterleaveBuffer(buffer_pointers);

  for (unsigned j = 0; j < buffers.size(); j++) {
    auto& buffer = buffers[j];
    assert(buffer.getFrameCount() == a1.getFrameCount());
    for (unsigned i = 0; i < buffer.size(); i++) {
      assert(buffer[i] == (i > buffer.size() / 2) ? 5 : 1);
    }
  }

  a1.interleaveBuffers(buffer_pointers);
  assert(a1.getFrameCount() == buffers[0].getFrameCount());
  for (unsigned i = 0; i < a1.size(); i++) {
    assert(a1[i] == (i > a1.size() / 2) ? 5 : 1);
  }
}

int main() {
  testConstructors();
  testResizing();
  testInterleavingStereo();
```

```
  testInterleavingMono();
  return 0;
}
#include "tsal.hpp"
#include <cassert>
#include <omp.h>

using namespace tsal;

void testEffectChain() {
  unsigned currentTest = 0;
  Mixer mixer;
  Channel channel;

  std::cout << "Testing effect chain: " << std::endl;
  mixer.add(channel);

  // Add a bunch of effects around the same time
  omp_set_num_threads(50);
  #pragma omp parallel
  {
    Compressor compressor;
    channel.add(compressor);
  }
  std::cout << ++currentTest << std::flush;

  std::vector<Compressor> compressors(3, Compressor());
  channel.add(compressors[0]);
  channel.add(compressors[1]);
  channel.add(compressors[2]);
  assert(channel.getEffectCount() == 3);
  std::cout << ++currentTest << std::flush;

  channel.remove(compressors[1]);
  assert(channel.getEffectCount() == 2);
  std::cout << ++currentTest << std::flush;
}

void testCompressor() {
  Mixer mixer;
  Synth synth;
  Compressor compressor;
  mixer.add(compressor);
```

```
  mixer.add(synth);

  // Test parameter setting
  compressor.setAttackTime(-1.0);
  compressor.setAttackTime(3000.0);
  compressor.setAttackTime(1000.0);

  compressor.setReleaseTime(-1.0);
  compressor.setReleaseTime(3000.0);
  compressor.setReleaseTime(1000.0);

  compressor.setThreshold(-40.0);
  compressor.setThreshold(500.0);
  compressor.setThreshold(60.0);

  compressor.setRatio(0.0);
  compressor.setRatio(30.0);
  compressor.setRatio(2.0);

  compressor.setPreGain(-40.0);
  compressor.setPreGain(40.0);
  compressor.setPreGain(0.0);

  compressor.setPostGain(-40.0);
  compressor.setPostGain(40.0);
  compressor.setPostGain(0.0);

}

void testDelay() {
  Mixer mixer;
  Synth synth;
  Delay delay;
  mixer.add(delay);
  mixer.add(synth);
}

int main() {
  testEffectChain();
  testCompressor();
  testDelay();
  return 0;
}
```

```
add_executable (effects effects.cpp)
add_executable (mixer mixer.cpp)
add_executable (localtest test.cpp)

target_link_libraries(effects tsal)
target_link_libraries(mixer tsal)
target_link_libraries(localtest tsal)

enable_testing()
add_test(MyEffectsTest effects)
add_test(MyMixerTest mixer)
add_test(MyTest localtest)
#include "tsal.hpp"
#include <cassert>

using namespace tsal;

void testConstructors() {
  Mixer mixer;
  Mixer mixer2(44100);

  Synth synth;
  mixer.add(synth);
  mixer2.add(synth);
  mixer2.remove(synth);
  mixer.add(synth);
}

void testAddRemove() {
  Mixer mixer;
  Channel channel;
  Synth synth[2]{Synth(), Synth()};
  Delay delay;
  mixer.add(channel);

  mixer.add(synth[0]);
  mixer.add(synth[1]);

  mixer.remove(synth[0]);
  mixer.remove(synth[1]);
  channel.add(synth[0]);

  mixer.add(delay);
```

```
  mixer.remove(delay);
  mixer.remove(channel);
}


int main() {
  testConstructors();
  testAddRemove();
  return 0;
}

#include "Oscillator.hpp"
#include "tsal.hpp"
#include <omp.h>
#include <ladspa.h>

using namespace tsal;

int main() {
  Mixer mixer;
  Synth synth;
  synth.setMode(Oscillator::SAW);

  Delay delay;
  /*
  LadspaEffect effect;
  LadspaEffect effect2;
  effect.loadPlugin("/home/mark/Downloads/ladspa_sdk_1.15/plugins/filter.so");
  effect2.loadPlugin("/home/mark/Downloads/ladspa_sdk_1.15/plugins/amp.so");
  mixer.add(effect);
  mixer.add(effect2);
  */


  mixer.add(synth);
  // mixer.add(delay);
  synth.play(tsal::C4);
  tsal::Util::thread_sleep(2000);
  synth.stop();
  tsal::Util::thread_sleep(2000);
  exit (0);
}
```

```
#! /bin/sh

# audiobuffer - temporary wrapper script for .libs/audiobuffer
# Generated by libtool (GNU libtool) 2.4.6
#
# The audiobuffer program cannot be directly executed until all the libtool
# libraries that it depends on are installed.
#
# This wrapper script should never be moved out of the build directory.
# If it is, it will not operate correctly.

# Sed substitution that helps us do robust quoting.  It backslashifies
# metacharacters that are still active within double-quoted strings.
sed_quote_subst='s|\([`"$\\]\)|\\\1|g'

# Be Bourne compatible
if test -n "${ZSH_VERSION+set}" && (emulate sh) >/dev/null 2>&1; then
  emulate sh
  NULLCMD=:
  # Zsh 3.x and 4.x performs word splitting on ${1+"$@"}, which
  # is contrary to our usage.  Disable this feature.
  alias -g '${1+"$@"}'='"$@"'
  setopt NO_GLOB_SUBST
else
  case `(set -o) 2>/dev/null` in *posix*) set -o posix;; esac
fi
BIN_SH=xpg4; export BIN_SH # for Tru64
DUALCASE=1; export DUALCASE # for MKS sh

# The HP-UX ksh and POSIX shell print the target directory to stdout
# if CDPATH is set.
(unset CDPATH) >/dev/null 2>&1 && unset CDPATH

relink_command=""

# This environment variable determines our operation mode.
if test "$libtool_install_magic" = "%%%MAGIC variable%%%"; then
  # install mode needs the following variables:
  generated_by_libtool_version='2.4.6'
  notinst_deplibs=' ../src/libtsal.la'
else
  # When we are sourced in execute mode, $file and $ECHO are already set.
  if test "$libtool_execute_magic" != "%%%MAGIC variable%%%"; then
```

```
    file="$0"

# A function that is used when there is no print builtin or printf.
func_fallback_echo ()
{
  eval 'cat <<_LTECHO_EOF
$1
_LTECHO_EOF'
}
    ECHO="printf %s\\n"
  fi

# Very basic option parsing. These options are (a) specific to
# the libtool wrapper, (b) are identical between the wrapper
# /script/ and the wrapper /executable/ that is used only on
# windows platforms, and (c) all begin with the string --lt-
# (application programs are unlikely to have options that match
# this pattern).
#
# There are only two supported options: --lt-debug and
# --lt-dump-script. There is, deliberately, no --lt-help.
#
# The first argument to this parsing function should be the
# script's ../libtool value, followed by no.
lt_option_debug=
func_parse_lt_options ()
{
  lt_script_arg0=$0
  shift
  for lt_opt
  do
    case "$lt_opt" in
    --lt-debug) lt_option_debug=1 ;;
    --lt-dump-script)
        lt_dump_D=`$ECHO "X$lt_script_arg0" | /usr/bin/sed -e 's/^X//' -e 's%/[^/]*$%%'`
        test "X$lt_dump_D" = "X$lt_script_arg0" && lt_dump_D=.
        lt_dump_F=`$ECHO "X$lt_script_arg0" | /usr/bin/sed -e 's/^X//' -e 's%^.*/%%'`
        cat "$lt_dump_D/$lt_dump_F"
        exit 0
      ;;
    --lt-*)
        $ECHO "Unrecognized --lt- option: '$lt_opt'" 1>&2
        exit 1
```

```
    ;;
  esac
 done

 # Print the debug banner immediately:
 if test -n "$lt_option_debug"; then
  echo "audiobuffer:audiobuffer:$LINENO: libtool wrapper (GNU libtool) 2.4.6" 1>&2
 fi
}

# Used when --lt-debug. Prints its arguments to stdout
# (redirection is the responsibility of the caller)
func_lt_dump_args ()
{
 lt_dump_args_N=1;
 for lt_arg
 do
  $ECHO "audiobuffer:audiobuffer:$LINENO: newargv[$lt_dump_args_N]: $lt_arg"
  lt_dump_args_N=`expr $lt_dump_args_N + 1`
 done
}

# Core function for launching the target application
func_exec_program_core ()
{

    if test -n "$lt_option_debug"; then
      $ECHO "audiobuffer:audiobuffer:$LINENO: newargv[0]: $progdir/$program" 1>&2
      func_lt_dump_args ${1+"$@"} 1>&2
    fi
    exec "$progdir/$program" ${1+"$@"}

    $ECHO "$0: cannot exec $program $*" 1>&2
    exit 1
}

# A function to encapsulate launching the target application
# Strips options in the --lt-* namespace from $@ and
# launches target application with the remaining arguments.
func_exec_program ()
{
 case " $* " in
 *\ --lt-*)
```

```
    for lt_wr_arg
    do
      case $lt_wr_arg in
      --lt-*) ;;
      *) set x "$@" "$lt_wr_arg"; shift;;
      esac
      shift
    done ;;
  esac
  func_exec_program_core ${1+"$@"}
}

  # Parse options
  func_parse_lt_options "$0" ${1+"$@"}

  # Find the directory that this script lives in.
  thisdir=`$ECHO "$file" | /usr/bin/sed 's%/[^/]*$%%'`
  test "x$thisdir" = "x$file" && thisdir=.

  # Follow symbolic links until we get to the real thisdir.
  file=`ls -ld "$file" | /usr/bin/sed -n 's/.*-> //p'`
  while test -n "$file"; do
    destdir=`$ECHO "$file" | /usr/bin/sed 's%/[^/]*$%%'`

    # If there was a directory component, then change thisdir.
    if test "x$destdir" != "x$file"; then
      case "$destdir" in
      [\\/]* | [A-Za-z]:[\\/]*) thisdir="$destdir" ;;
      *) thisdir="$thisdir/$destdir" ;;
      esac
    fi

    file=`$ECHO "$file" | /usr/bin/sed 's%^.*/%%'`
    file=`ls -ld "$thisdir/$file" | /usr/bin/sed -n 's/.*-> //p'`
  done

  # Usually 'no', except on cygwin/mingw when embedded into
  # the cwrapper.
  WRAPPER_SCRIPT_BELONGS_IN_OBJDIR=no
  if test "$WRAPPER_SCRIPT_BELONGS_IN_OBJDIR" = "yes"; then
    # special case for '.'
    if test "$thisdir" = "."; then
      thisdir=`pwd`
```

```
  fi
  # remove .libs from thisdir
  case "$thisdir" in
  *[\\/].libs ) thisdir=`$ECHO "$thisdir" | /usr/bin/sed 's%[\\/][^\\/]*$%%'` ;;
  .libs )   thisdir=. ;;
  esac
 fi

 # Try to get the absolute directory name.
 absdir=`cd "$thisdir" && pwd`
 test -n "$absdir" && thisdir="$absdir"

 program='audiobuffer'
 progdir="$thisdir/.libs"


 if test -f "$progdir/$program"; then
  # Add our own library path to LD_LIBRARY_PATH
  LD_LIBRARY_PATH="/home/mark/Programming/TSAL/src/.libs:$LD_LIBRARY_PATH"

  # Some systems cannot cope with colon-terminated LD_LIBRARY_PATH
  # The second colon is a workaround for a bug in BeOS R4 sed
  LD_LIBRARY_PATH=`$ECHO "$LD_LIBRARY_PATH" | /usr/bin/sed 's/::*$//'`

  export LD_LIBRARY_PATH

  if test "$libtool_execute_magic" != "%%%MAGIC variable%%%"; then
   # Run the actual program with our arguments.
   func_exec_program ${1+"$@"}
  fi
 else
  # The program doesn't exist.
  $ECHO "$0: error: '$progdir/$program' does not exist" 1>&2
  $ECHO "This script is just a wrapper for $program." 1>&2
  $ECHO "See the libtool documentation for more information." 1>&2
  exit 1
 fi
fi
# - Try to find Portaudio
# Once done this will define
#
#  PORTAUDIO_FOUND - system has Portaudio
#  PORTAUDIO_INCLUDE_DIRS - the Portaudio include directory
```

```
#  PORTAUDIO_LIBRARIES - Link these to use Portaudio
#  PORTAUDIO_DEFINITIONS - Compiler switches required for using Portaudio
#  PORTAUDIO_VERSION - Portaudio version
#
#  Copyright (c) 2006 Andreas Schneider <mail@cynapses.org>
#
# Redistribution and use is allowed according to the terms of the New BSD license.
# For details see the accompanying COPYING-CMAKE-SCRIPTS file.
#


if (PORTAUDIO_LIBRARIES AND PORTAUDIO_INCLUDE_DIRS)
  # in cache already
  set(PORTAUDIO_FOUND TRUE)
else (PORTAUDIO_LIBRARIES AND PORTAUDIO_INCLUDE_DIRS)
  if (NOT WIN32)
   include(FindPkgConfig)
   pkg_check_modules(PORTAUDIO2 portaudio-2.0)
  endif (NOT WIN32)

  if (PORTAUDIO2_FOUND)
    set(PORTAUDIO_INCLUDE_DIRS
      ${PORTAUDIO2_INCLUDE_DIRS}
    )
    if (${CMAKE_SYSTEM_NAME} MATCHES "Darwin")
      set(PORTAUDIO_LIBRARIES
"${PORTAUDIO2_LIBRARY_DIRS}/lib${PORTAUDIO2_LIBRARIES}.dylib")
    else (${CMAKE_SYSTEM_NAME} MATCHES "Darwin")
      set(PORTAUDIO_LIBRARIES
        ${PORTAUDIO2_LIBRARIES}
      )
    endif (${CMAKE_SYSTEM_NAME} MATCHES "Darwin")
    set(PORTAUDIO_VERSION
      19
    )
    set(PORTAUDIO_FOUND TRUE)
  else (PORTAUDIO2_FOUND)
    find_path(PORTAUDIO_INCLUDE_DIR
      NAMES
        portaudio.h
      PATHS
        /usr/include
        /usr/local/include
```

```
    /opt/local/include
    /sw/include
)

find_library(PORTAUDIO_LIBRARY
  NAMES
    portaudio
  PATHS
    /usr/lib
    /usr/local/lib
    /opt/local/lib
    /sw/lib
    ${PORTAUDIO_LIBRARY_DIR}
)

set(PORTAUDIO_INCLUDE_DIRS
  ${PORTAUDIO_INCLUDE_DIR}
)
set(PORTAUDIO_LIBRARIES
  ${PORTAUDIO_LIBRARY}
)

set(PORTAUDIO_LIBRARY_DIRS
  ${PORTAUDIO_LIBRARY_DIR}
)

set(PORTAUDIO_VERSION
  18
)

if (PORTAUDIO_INCLUDE_DIRS AND PORTAUDIO_LIBRARIES)
   set(PORTAUDIO_FOUND TRUE)
endif (PORTAUDIO_INCLUDE_DIRS AND PORTAUDIO_LIBRARIES)

if (PORTAUDIO_FOUND)
  if (NOT Portaudio_FIND_QUIETLY)
    message(STATUS "Found Portaudio: ${PORTAUDIO_LIBRARIES}")
  endif (NOT Portaudio_FIND_QUIETLY)
else (PORTAUDIO_FOUND)
  if (Portaudio_FIND_REQUIRED)
    message(FATAL_ERROR "Could not find Portaudio")
  endif (Portaudio_FIND_REQUIRED)
endif (PORTAUDIO_FOUND)
```

```
  endif (PORTAUDIO2_FOUND)


  # show the PORTAUDIO_INCLUDE_DIRS and PORTAUDIO_LIBRARIES variables only in
the advanced view
  mark_as_advanced(PORTAUDIO_INCLUDE_DIRS PORTAUDIO_LIBRARIES)

endif (PORTAUDIO_LIBRARIES AND PORTAUDIO_INCLUDE_DIRS)
@PACKAGE_INIT@

find_dependency(Boost 1.60 REQUIRED COMPONENTS regex)
include(${PROJECT_SOURCE_DIR}/src/tsalTargets.cmake)
macro(FetchContent_MakeAvailable NAME)
   FetchContent_GetProperties(${NAME})
   if(NOT ${NAME}_POPULATED)
      FetchContent_Populate(${NAME})
      add_subdirectory(${${NAME}_SOURCE_DIR} ${${NAME}_BINARY_DIR})
   endif()
endmacro()
# Versioning
cmake_minimum_required(VERSION 3.10...3.15)

if(${CMAKE_VERSION} VERSION_LESS 3.12)
   cmake_policy(VERSION ${CMAKE_MAJOR_VERSION}.${CMAKE_MINOR_VERSION})
endif()

# Variables so we don't repeat ourselves
set(LOCAL_PROJECT_VERSION     "1.0.0")
set(LOCAL_PROJECT_VENDOR      "Calvin University")
set(LOCAL_PROJECT_NAMESPACE   "tsal")
set(LOCAL_PROJECT_NAME        "tsal")
set(LOCAL_PROJECT_OUTPUT_NAME "tsal")
set(LOCAL_PROJECT_DESCRIPTION "Thread safe Autio Library")

set(CMAKE_INSTALL_PREFIX /usr/local/)

# Project details
project(${LOCAL_PROJECT_NAME}
      VERSION ${LOCAL_PROJECT_VERSION}
      DESCRIPTION ${LOCAL_PROJECT_DESCRIPTION}
      LANGUAGES C CXX)

set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
```

```
# detect operating system and host processor
message(STATUS "We are on a ${CMAKE_SYSTEM_NAME} system")
message(STATUS "The host processor is ${CMAKE_HOST_SYSTEM_PROCESSOR}")

option(INSTALL_HEADERS "Install library headers" ON)
option(BUILD_TESTS "Build unit tests" ON)

list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_SOURCE_DIR}/cmake")

include(CheckIncludeFiles)
check_include_files(stddef.h HAVE_STDDEF_H)
check_include_files(stdint.h HAVE_STDINT_H)
check_include_files(stdlib.h HAVE_STDLIB_H)
check_include_files(sys/time.h HAVE_SYSTIME_H)
check_include_files(unistd.h HAVE_UNISTD_H)

include(CheckFunctionExists)
check_function_exists(fseeko HAVE_FSEEKO)
check_function_exists(error_at_line HAVE_ERROR_AT_LINE)
check_function_exists(malloc HAVE_MALLOC)
check_function_exists(mbrtowc HAVE_MBRTOWC)
check_function_exists(mktime HAVE_MKTIME)
check_function_exists(realloc HAVE_REALLOC)
check_function_exists(strtod HAVE_STRTOD)
check_function_exists(atexit HAVE_ATEXIT)
check_function_exists(btowc HAVE_BTOWC)
check_function_exists(clock_gettime HAVE_CLOCK_GETTIME)
check_function_exists(getdelim HAVE_GETDELIM)
check_function_exists(isascii HAVE_ISASCII)
check_function_exists(iswprint HAVE_ISWPRINT)
check_function_exists(localeconv HAVE_LOCALECONV)
check_function_exists(localtime_r HAVE_LOCALTIME_R)
check_function_exists(mblen HAVE_MBLEN)
check_function_exists(mbrlen HAVE_MBRLEN)
check_function_exists(pow HAVE_POW)
check_function_exists(putenv HAVE_PUTENV)
check_function_exists(realpath HAVE_REALPATH)
check_function_exists(rpmatch HAVE_RPMATCH)
check_function_exists(select HAVE_SELECT)
check_function_exists(setenv HAVE_SETENV)
check_function_exists(setlocale HAVE_SETLOCALE)
check_function_exists(stime HAVE_STIME)
check_function_exists(strtol HAVE_STRTOL)
```

```
check_function_exists(strtoul HAVE_STROUL)
check_function_exists(strtoull HAVE_STROULL)
check_function_exists(tzset HAVE_TZSET)
check_function_exists(wcwidth HAVE_WCWIDTH)
check_function_exists(obstack HAVE_OBSTACK)

include(CheckTypeSize)
# # Check for integer types
check_type_size("short" SIZE_OF_SHORT)
check_type_size("int" SIZE_OF_INT)
check_type_size("long" SIZE_OF_LONG)
check_type_size("long long" SIZE_OF_LONG_LONG)

check_type_size("unsigned short" SIZE_OF_UNSIGNED_SHORT)
check_type_size("unsigned" SIZE_OF_UNSIGNED)
check_type_size("unsigned long" SIZE_OF_UNSIGNED_LONG)
check_type_size("unsigned long long" SIZE_OF_UNSIGNED_LONG_LONG)

check_type_size("__int64" __INT64)
check_type_size("unsigned __int64" UNSIGNED___INT64)

check_type_size(int16_t INT16_T)
check_type_size(int32_t INT32_T)
check_type_size(int64_t INT64_T)
check_type_size(intmax_t INTMAX_T)
check_type_size(uint8_t UINT8_T)
check_type_size(uint16_t UINT16_T)
check_type_size(uint32_t UINT32_T)
check_type_size(uint64_t UINT64_T)
check_type_size(uintmax_t UINTMAX_T)

check_type_size(size_t SIZE_T)
if(NOT HAVE_SIZE_T)
  if("${CMAKE_SIZEOF_VOID_P}" EQUAL 8)
    set(size_t "uint64_t")
  else("${CMAKE_SIZEOF_VOID_P}" EQUAL 8)
    set(size_t   "uint32_t")
  endif("${CMAKE_SIZEOF_VOID_P}" EQUAL 8)
endif(NOT HAVE_SIZE_T)

check_type_size(ssize_t SSIZE_T)
if(NOT HAVE_SSIZE_T)
  if("${CMAKE_SIZEOF_VOID_P}" EQUAL 8)
```

```
    set(ssize_t "int64_t")
  else("${CMAKE_SIZEOF_VOID_P}" EQUAL 8)
    set(ssize_t "long")
  endif("${CMAKE_SIZEOF_VOID_P}" EQUAL 8)
endif(NOT HAVE_SSIZE_T)

# Only do these if this is the main project, and not if it is included through
add_subdirectory
if(CMAKE_PROJECT_NAME STREQUAL PROJECT_NAME)
   # Optionally set things like CMAKE_CXX_STANDARD,
CMAKE_POSITION_INDEPENDENT_CODE here

   if(CMAKE_CXX_STANDARD EQUAL "98" )
     message(FATAL_ERROR "CMAKE_CXX_STANDARD:STRING=98 is not supported.")
   endif()

   if(NOT CMAKE_CXX_STANDARD)
     set(CMAKE_CXX_STANDARD 20) # Supported values are ``11``, ``14``, and ``17``.
   endif()
   if(NOT CMAKE_CXX_STANDARD_REQUIRED)
     set(CMAKE_CXX_STANDARD_REQUIRED ON)
   endif()
   if(NOT CMAKE_CXX_EXTENSIONS)
     set(CMAKE_CXX_EXTENSIONS OFF)
   endif()

   # Enable runtime search path support for dynamic libraries on OSX
   if(APPLE)
     set(CMAKE_MACOSX_RPATH 1)
   endif()

   # Let's nicely support folders in IDE's
   set_property(GLOBAL PROPERTY USE_FOLDERS ON)

   include(GNUInstallDirs)

   # Testing only available if this is the main app
   # Note this needs to be done in the main CMakeLists
   # since it calls enable_testing, which must be in the
   # main CMakeLists.
   include(CTest)

   # Docs only available if this is the main app
```

```
  find_package(Doxygen)
  if(Doxygen_FOUND)
   set(DOXYGEN_GENERATE_HTML YES)
   set(DOXYGEN_EXCLUDE_PATTERNS
    */.git/*
    */.svn/*
    */.hg/*
    */CMakeFiles/*
    */_CPack_Packages/*
    DartConfiguration.tcl
    CMakeLists.txt
    CMakeCache.txt
    */extern/*
   )
   message(STATUS "Working on Doxygen")
   doxygen_add_docs(
    doxygen
    ${PROJECT_SOURCE_DIR}
    COMMENT "Generate man pages"
   )
   message(STATUS "Generated")
  else()
  message(STATUS "Doxygen not found, not building docs")
  endif()
endif()

# Git sumodule auto build
find_package(Git QUIET)
if(GIT_FOUND AND EXISTS "${PROJECT_SOURCE_DIR}/.git")
# Update submodules as needed
  option(GIT_SUBMODULE "Check submodules during build" ON)
  if(GIT_SUBMODULE)
    message(STATUS "Submodule update")
    execute_process(COMMAND ${GIT_EXECUTABLE} submodule update --init
--recursive
                    WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
                    RESULT_VARIABLE GIT_SUBMOD_RESULT)
    if(NOT GIT_SUBMOD_RESULT EQUAL "0")
      message(FATAL_ERROR "git submodule update --init failed with
${GIT_SUBMOD_RESULT}, please checkout submodules")
    endif()
  endif()
endif()
```

```
if(NOT EXISTS "${PROJECT_SOURCE_DIR}/extern/midifile")
   message(FATAL_ERROR "The submodules were not downloaded! GIT_SUBMODULE
was turned off or failed. Please update submodules and try again.")
endif()

if(NOT EXISTS "${PROJECT_SOURCE_DIR}/extern/TinySoundFont")
   message(FATAL_ERROR "The submodules were not downloaded! GIT_SUBMODULE
was turned off or failed. Please update submodules and try again.")
endif()

if(NOT EXISTS "${PROJECT_SOURCE_DIR}/extern/portaudio")
   message(FATAL_ERROR "The submodules were not downloaded! GIT_SUBMODULE
was turned off or failed. Please update submodules and try again.")
endif()

# System Dependencies
# find_package(Portaudio REQUIRED)

find_package(Threads REQUIRED)
set(THREADS_PREFER_PTHREAD_FLAG ON)

find_package(OpenMP REQUIRED)

set(LIBRARY_OUTPUT_PATH "${CMAKE_BINARY_DIR}")
set(EXECUTABLE_OUTPUT_PATH "${CMAKE_BINARY_DIR}")

add_subdirectory(extern)

# The compiled library code is here
add_subdirectory(src)

# The example code is here
add_subdirectory(examples)

# Testing only available if this is the main app
if((CMAKE_PROJECT_NAME STREQUAL PROJECT_NAME OR
MODERN_CMAKE_BUILD_TESTING) AND BUILD_TESTING)
   add_subdirectory(tests)
endif()

set(CPACK_GENERATOR "DEB")
set(CPACK_DEBIAN_PACKAGE_MAINTAINER "Mark Wissink")
```

```
set(CPACK_PACKAGE_VERSION ${LOCAL_PROJECT_VERSION})
include(CPack)
#include "tsal.hpp"
#include <pthread.h>
#include <vector>

// Temporary fix
#include <chrono>
#include <thread>
#define NUM_THREADS 3

void Util::thread_sleep(unsigned milliseconds) {
  std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds));
}
// Temporary fix


struct ThreadData {
  tsal::MidiParser* midiParser;
  std::vector<tsal::Oscillator>* voices;
  unsigned tid;
};

void* ThreadFunction(void* ptr) {
  ThreadData data = *((ThreadData*) ptr);
  auto& voices = *data.voices;
  auto& midiParser = *data.midiParser;

  Util::thread_sleep(data.tid * midiParser.quaterNoteMs(4));
  voices[data.tid].setActive();

  for (unsigned i = 0; i < midiParser.size() - 1; i++) {
    auto& me = midiParser[i];
    if (me.isNoteOn())
      voices[data.tid].playNote(me.getKeyNumber(), me.getVelocity());

    Util::thread_sleep(midiParser.ticksToMs(midiParser[i + 1].tick - me.tick));
    voices[data.tid].stop();
  }
  pthread_exit(NULL);
}

int main(int argc, char* argv[]) {
```

```
  if (argc != 3) {
    std::cout << "Invalid arguments\n\n"
              << "jubilate <midifile> <voices>\n"
              << "\tmidifile = a path to a midifile\n"
              << "\tvoices = the number of voices (<6)\n"
              << std::endl;
    return 0;
  }
  tsal::MidiParser midiParser(1, argv[1]);
  // const unsigned numVoices = atoi(argv[2]);

  pthread_t threads[NUM_THREADS];
  ThreadData* threadData[NUM_THREADS];
  tsal::Mixer mixer;
  std::vector<tsal::Oscillator> voices(NUM_THREADS);

  for (unsigned i = 0; i < voices.size(); i++) {
    voices[i].setGain(-20);
    voices[i].setActive(false);
    mixer.add(voices[i]);
  }

  for(unsigned i = 0; i < NUM_THREADS; i++ ) {
    threadData[i] = new ThreadData { &midiParser, &voices, i };
    pthread_create(&threads[i], NULL, ThreadFunction, (void *) threadData[i]);
  }

  for (unsigned i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
    delete threadData[i];
  }

  return 0;
}
#include "tsal.hpp"
#include <omp.h>
#include <vector>

/** @example canon.cpp
 *
 * ThreadSynth is an instrument that uses blocking calls to schedule its midi events.
 * If this instrument was used in a process with only one thread, there could only be
 * one synth playing at a time. To achieve a multi-instrument sound, each ThreadSynth
```

```
 * will need its own thread. The omp parallel pragma creates these threads for each synth
 * so that when the ThreadSynth makes a call to play or stop, only that synth's thread
 * is blocked
 *
 * A canon (or a round) is a musical composition technique that takes a melody and
duplicates it over multiple voices with some variation in time offset or pitch.
 * In this example, the given midifile is played on the specified number of threads, each
with a measure offset
 *
 * Jubilate Deo is a basic round
 * https://youtu.be/lLLvDfsl_iM?t=33 (sine waves sounds less angelic)
 *
 * Parse the parameters\n
 * Create the mixer and synths\n
 * Set the number of threads\n
 * Run parallel block
 * - Calculate an offset into the song for a round effect
 * - Play the midi events
 */
int main(int argc, char* argv[]) {
  if (argc != 3) {
    std::cout << "Invalid arguments\n\n"
              << "canon <midifile> <voices>\n"
              << "\tmidifile = a path to a midifile\n"
              << "\tvoices = the number of voices (<6)\n"
              << std::endl;
    return 0;
  }
  tsal::MidiParser midiParser(1, argv[1]);
  const int numVoices = atoi(argv[2]);


  tsal::Mixer mixer;
  std::vector<tsal::ThreadSynth> voices(numVoices,
tsal::ThreadSynth(mixer.getContext()));

  omp_set_num_threads(numVoices);

  #pragma omp parallel
  {
    int id = omp_get_thread_num();

    mixer.add(voices[id]);
```

```
    voices[id].setVolume(0.5);

    // Wait to come in based on thread id
    voices[id].stop(tsal::Timing::HALF, id);

    for (unsigned i = 0; i < midiParser.size(); i++) {
      auto& me = midiParser[i];
      if (me.isNoteOn()) {
        voices[id].play(me.getKeyNumber(), tsal::Timing::TICK, me.getTickDuration());
      }
    }
  }
  return 0;
}
#include "tsal.hpp"
#include <thread>
#include <condition_variable>
#include <iostream>
#include <queue>

using namespace tsal;

bool run = true;

class SharedQueue {
  public:
    void produce(Synth* item) {
      std::unique_lock<std::mutex> lk(mMutex);
      mCondition.wait(lk, [this]{return mQueue.size() < mMaxQueueSize;});

      for (int i = 0; i < 100; i++) {
        item->play(item->getNote() + 0.25);
        Util::thread_sleep(5);
      }
      item->setVolume(0.5);
      Util::thread_sleep(500);

      mQueue.push(item);
      lk.unlock();
      mCondition.notify_all();
    };
```

```
    Synth* consume() {
      std::unique_lock<std::mutex> lk(mMutex);
      mCondition.wait(lk, [this]{return mQueue.size() > 0;});
      Synth* front = mQueue.front();

      front->setVolume(1);
      for (int i = 0; i < 100; i++) {
        front->play(front->getNote() + 0.25);
        Util::thread_sleep(5);
      }
      front->stop();

      mQueue.pop();
      lk.unlock();
      mCondition.notify_all();
      return front;
    };
  private:
    std::condition_variable mCondition;
    std::mutex mMutex;
    std::queue<Synth*> mQueue;
    unsigned mMaxQueueSize = 12;
};

void produce(SharedQueue* queue, Mixer* mixer) {
  Synth* item = nullptr;
  while (run) {
    Util::thread_sleep(rand() % 5000);
    item = new Synth(mixer);
    mixer->add(*item);
    item->setEnvelopeActive(false);
    item->setMode(Oscillator::SINE);
    item->play(C3);

    queue->produce(item);
  }
}

void consume(SharedQueue* queue, Mixer* mixer) {
  Synth* item = nullptr;
  while (run) {
    Util::thread_sleep(rand() % 5000);
    item = queue->consume();
```

```
    mixer->remove(*item);
    delete item;
  }
}

/** @example producer_consumer.cpp
 *
 * Producer and consumer is a common example of synchonization between multiple processes or threads.
 * The SharedQueue class acts as a monitor between the producers and consumers. The sine wave
 * pitch corresponds to the fullness of the queue, the square wave blips correspond to a production,
 * and the saw wave blips correspond to consumption
 *
 * Parse the parameter\n
 * Create the mixer and the SharedQueue
 * Start the producer and consumer threads
 * Producer threads:
 * - Make a random item
 * - Add the item to the queue
 * - Play a square wave with pitch based on item
 * Consumer threads:
 * - Consume an item from the queue
 * - Play a saw wave with pitch based on item
 * SharedQueue:
 * - Provide the synchronized access to the underlying queue
 * - Change the pitch of the sine wave whenever a item is added or removed
 */
int main(int argc, char* argv[]) {
  if (argc != 3) {
    std::cout << "Invalid arguments\n\n"
              << "producer_consumer <producers> <consumers>\n"
              << "\tproducers = the number of producers\n"
              << "\tconsumers = the number of consumers\n"
              << std::endl;
    return 0;
  }
  const int numProducers = atoi(argv[1]);
  const int numConsumers = atoi(argv[2]);
  Mixer mixer;
  SharedQueue queue;
```

```
  std::thread producers[numProducers];
  std::thread consumers[numConsumers];
  for (int i = 0; i < numProducers; i++) {
    producers[i] = std::thread(produce, &queue, &mixer);
  }
  for (int i = 0; i < numConsumers; i++) {
    consumers[i] = std::thread(consume, &queue, &mixer);
  }

  // Wait for the user to stop the synth
  char input;
  std::cout << "Press <enter> to quit:" << std::flush;
  std::cin.get(input);
  run = false;
  std::cout << "Quitting..." << std::endl;

  for (int i = 0; i < numProducers; i++) {
    producers[i].join();
  }
  for (int i = 0; i < numConsumers; i++) {
    consumers[i].join();
  }

  return 0;
}
#include "tsal.hpp"
#include <omp.h>

using namespace tsal;

#define MAX_VALUE 100000

enum MergeState {
  S_MERGE = 1,
  S_SHIFT = 2,
  S_WAIT = 3,
  S_DONE = 4,
  S_HIDE = 5
};

struct sortData {
  MergeState state;           //Current state of the threads
```

```
  int first, last,           //Start and end of our block
    left, right,             //Indices of two numbers to compare
    fi, hi, li,              //Indices of first middle and last numbers in a set
    depth;                   //Current depth of the merge
  int* a;                    //Array of numbers to sort
  int seg, segs;             //Current / total segments
  int size;

  sortData(int* arr, int f, int l) {
    fi = hi = li = 0;        //Initialize indices
    left = right = 0;        //Initialize bounds
    a = arr;                 //Get a pointer to the array we'll be sorting
    first = f;               //Set the first element we need to worry about
    last = l;                //Set the last element we need to worry about
    depth = 0;               //We start at depth 0
    seg = 0; segs = 1;       //We start on segment -1, with a total of 1 segment
    while(segs < (l-f)) {    //If the current number of segments is more than the # of
elements, we're done
      ++depth;               //Otherwise, increment the depth...
      segs *= 2;             //...and double the number of segments
    }
    state = S_SHIFT;         //Start Merging
    size = 2;
  }

  void restart(int l) {
    depth = 0;
    hi = last;
    right = hi+1;
    last = li = l;
    fi = left = first;
    state = S_MERGE;
    size *= 2;
  }

  void sortStep() {
    int tmp, pivot, jump;
    switch(state) {
      case S_SHIFT:
        pivot = jump = segs/2;
        fi = first; li = last;
        hi = (fi + li) / 2;    //Set our half index to the median of our first and last
        for (tmp = depth; tmp > 0; --tmp) {
```

```
        jump /= 2;
        if (seg < pivot) {
          pivot -= jump;
          li = hi;          //Set out last index to our old half index
        } else {
          pivot += jump;
          fi = hi+1;        //Set out first index to our old half index plus one
        }
        hi = (fi + li) / 2;  //Set our new half index to the median of our first and last
      }
      left = fi; right = hi+1;
      state = S_MERGE;          //We're ready to start Merging
      break;
    case S_MERGE:
      if (left > right || right > last) {
        seg = 0;               //Reset our segment(s)
        segs /= 2;              //We're now using half as many segments
        state = (depth-- == 0) ? S_WAIT : S_SHIFT;
      } else if (right > li) {
        ++seg; state = S_SHIFT;  //Move on to the next segment and recalculate our first
and last indices
      } else if (left <= hi && a[left] < a[right]) {
        ++left;
      } else {
        tmp = a[right];
        for (int x = right; x > left; --x)
          a[x] = a[x-1];
        a[left] = tmp;
        ++left; ++right; ++hi;
      }
      break;
    default:
      break;
  }
 }
};



/*!
 * \brief Visualization of the bottom-up mergesort algorithm.
 * \details Utilizes the sortData struct and sorts a number of items using the mergesort
algorithm.
```

```
 * \details Uses lines to represent the items being sorted.
 * \details At the start, the items being sorted are all divided.
 * \details Once items have been sorted in one divided section, then sections are merged
and the process repeats itself.
 * \details Different colors represent different sections being sorted.
 * \details Once all items have been sorted and merged, the animation stops and all lines
are colored white.
 */
void mergeSortFunction(std::vector<ThreadSynth>& voices, int threads, int size) {
  const int IPF = 1;      // Iterations per frame
  const int maxNumber = 100000;
  int* numbers = new int[size];       // Array to store the data
  for (int i = 0; i < size; i++)
    numbers[i] = rand() % maxNumber;

  int bs = size / threads;
  int ex = size % threads;
  sortData** sd = new sortData*[threads];
  int f = 0;
  int l = (ex == 0) ? bs-1 : bs;
  for (int i = 0; i < threads; ++i) {
    sd[i] = new sortData(numbers,f,l);
    f = l+1;
    if (i < ex-1) l += (bs + 1);
    else          l += bs;
  }
  #pragma omp parallel num_threads(threads)
  {
    int tid = omp_get_thread_num();
    auto& voice = voices[tid];
    //std::cout << tid << std::endl;
    while (true) {
      if (sd[tid]->state == S_WAIT) {  //Merge waiting threads
        voice.stop();

        if ((tid % sd[tid]->size) > 0) {
          sd[tid]->state = S_DONE;
        } else {
          int next = tid+sd[tid]->size/2;
          if (next < threads && sd[next]->state == S_DONE) {
            sd[next]->state = S_HIDE;

            sd[tid]->restart(sd[next]->last);
```

```
        }
       }
      }
      for (int i = 0; i < IPF; i++)
       sd[tid]->sortStep();

      double number;
      MergeState state = sd[tid]->state;
      if (state != S_HIDE && state != S_DONE) {
       for (int i = sd[tid]->first; i < sd[tid]->last; ++i) {
        number = numbers[i];
        // If we are processing the item, play a sound
        if (i == sd[tid]->left) {
         voice.play(C2 + (tid * 3) + 60 * (number / maxNumber), Timing::MICROSECOND,
50);
        }
       }
      }
     }
    }
    for (int i = 0; i < threads; ++i)
     delete sd[i];
    delete [] sd;
    delete [] numbers;
}

/** @example merge_sort.cpp
 *
 * Merge sort is a sorting algorithm that can be done in parallel. As a result,
 * each thread involved in the algorithm is assigned an oscillator which plays
 * the pitch of the processed item plus a constant. Each thread has its own base
 * pitch so you can differentiate between them near the end of the sorting process.
 *
 * Parse the parameters\n
 * Create the mixer and oscillators\n
 * Start the merge sort algorithm
 * For each thread:
 * - Process an item in the merge step and set the oscillator pitch accordingly
 * - When complete with job, mute the oscillator
 */
int main() {

   int threads, t = 1;
```

```
   for (threads = 1; threads < t; threads *=2);  //Force threads to be a power of 2

   Mixer mixer;
   std::vector<ThreadSynth> voices(threads, ThreadSynth());
   for (unsigned i = 0; i < voices.size(); i++) {
     mixer.add(voices[i]);
     voices[i].setEnvelopeActive(false);
   }
   mergeSortFunction(voices, threads, 5000);
}
#include "tsal.hpp"
#include <omp.h>
#include <vector>

#define NUM_THREADS 3

int main() {
  tsal::Mixer mixer;
  std::vector<tsal::Oscillator> voices(NUM_THREADS);

  tsal::MidiNote chord[] = {tsal::C4, tsal::E4, tsal::G4};
  for (unsigned i = 0; i < voices.size(); i++) {
    voices[i].setGain(-20);
    voices[i].setNote(chord[i]);
  }

  omp_set_num_threads(NUM_THREADS);

  #pragma omp parallel
  {
    int id = omp_get_thread_num();

    #pragma omp for
    for (unsigned i = 0; i < 100; i++) {
      voices[id].setFrequency(voices[id].getFrequency() + 10);
      tsal::Util::thread_sleep(100);
    }

  }
  return 0;
}
#include "tsal.hpp"
```

```
using namespace tsal;

#define MAX_VALUE 100000

void quickSort(ThreadSynth& synth, int* data, int low, int high) {
  if (low < high) {
    // Partition
    int pivotValue = data[low];
    int pivot = low;
    for (int i = low + 1; i < high; i++) {
      synth.play(C3 + 45 * ((double) data[i] / MAX_VALUE), Timing::MICROSECOND, 50);

      if (data[i] < pivotValue) {
        pivot++;
        std::swap(data[i], data[pivot]);
      }
    }
    std::swap(data[low], data[pivot]);

    quickSort(synth, data, low, pivot - 1);
    quickSort(synth, data, pivot + 1, high);
  }
}

int main() {
  Mixer mixer;
  ThreadSynth synth;
  mixer.add(synth);
  synth.setEnvelopeActive(false);

  // Generate the data
  const int size = 5000;
  int* data = new int[size];
  for (int i = 0; i < size; i++) {
    data[i] = rand() % MAX_VALUE;
  }

  // Sort the data
  quickSort(synth, data, 0, size);
}
#include "tsal.hpp"
#include <pthread.h>
#include <vector>
```

```
// Temporary fix
#include <chrono>
#include <thread>
#define NUM_THREADS 2

void Util::thread_sleep(unsigned milliseconds) {
  std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds));
}
// Temporary fix


struct ThreadData {
  tsal::MidiParser* midiParser;
  std::vector<tsal::Oscillator>* voices;
  unsigned tid;
  unsigned start;
  unsigned end;
};

void* ThreadFunction(void* ptr) {
  ThreadData data = *((ThreadData*) ptr);
  auto& voices = *data.voices;
  auto& midiParser = *data.midiParser;

  voices[data.tid].setActive();

  for (unsigned i = data.start; i < data.end; i++) {
    auto& me = midiParser[i];
    if (me.isNote())
      voices[data.tid].playNote(me.getKeyNumber(), me.getVelocity());

    Util::thread_sleep(midiParser.ticksToMs(midiParser[i + 1].tick - me.tick));
  }
  pthread_exit(NULL);
}


int main(int argc, char* argv[]) {
  if (argc != 3) {
    std::cout << "Invalid arguments\n\n"
              << "rain <midifile> <tracks>\n"
              << "\tmidifile = a path to a midifile\n"
```

```
            << "\ttracks = the number of tracks in the midifile\n"
            << std::endl;
    return 0;
  }
  const unsigned numTracks = atoi(argv[2]);
  tsal::MidiParser midiParser(numTracks, argv[1]);

  pthread_t threads[NUM_THREADS];
  ThreadData* threadData[NUM_THREADS];
  tsal::Mixer mixer;
  std::vector<tsal::Oscillator> voices(NUM_THREADS);

  for (unsigned i = 0; i < voices.size(); i++) {
    voices[i].setGain(-20);
    voices[i].setActive(false);
    mixer.add(voices[i]);
  }

  unsigned blockSize = midiParser.size() / numTracks;
  for(unsigned i = 0; i < NUM_THREADS; i++ ) {
    unsigned start = i * blockSize;
    unsigned end = start + blockSize;
    threadData[i] = new ThreadData { &midiParser, &voices, i, start, end };
    pthread_create(&threads[i], NULL, ThreadFunction, (void *) threadData[i]);
  }

  for (unsigned i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
    delete threadData[i];
  }

  return 0;
}
#include "tsal.hpp"
#include <pthread.h>
#include <vector>

// Temporary fix
#include <chrono>
#include <thread>
#define NUM_THREADS 3

void Util::thread_sleep(unsigned milliseconds) {
```

```
  std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds));
}
// Temporary fix


struct ThreadData {
  std::vector<tsal::Oscillator>* voices;
  unsigned tid;
  unsigned start;
  unsigned end;
};

void* ThreadFunction(void* ptr) {
  ThreadData data = *((ThreadData*) ptr);
  auto& voices = *data.voices;

  for (unsigned i = data.start; i < data.end; i++) {
    voices[data.tid].setFrequency(voices[data.tid].getFrequency() + 10);
    Util::thread_sleep(100);
  }
  pthread_exit(NULL);
}

int main() {
  pthread_t threads[NUM_THREADS];
  ThreadData* threadData[NUM_THREADS];
  tsal::Mixer mixer;
  std::vector<tsal::Oscillator> voices(NUM_THREADS);

  tsal::MidiNote chord[3] = {tsal::C4, tsal::E4, tsal::G4};
  for (unsigned i = 0; i < voices.size(); i++) {
    voices[i].setGain(-20);
    voices[i].setNote(chord[i]);
    mixer.add(voices[i]);
  }

  for(unsigned i = 0; i < NUM_THREADS; i++ ) {
    threadData[i] = new ThreadData { &voices, i, 0, 100/NUM_THREADS };
    pthread_create(&threads[i], NULL, ThreadFunction, (void *) threadData[i]);
  }

  for (unsigned i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
```

```
    delete threadData[i];
  }

  return 0;
}
# Add executable
add_executable(bubble_sort bubble_sort.cpp)
# add_executable(canon canon.cpp)
# add_executable(chord chord.cpp)
# add_executable(chord_p chord_p.cpp)
add_executable(hello_tsal hello_tsal.cpp)
add_executable(hello_tsal_threads hello_tsal_threads.cpp)
# add_executable(jubilate_p jubilate_p.cpp)
add_executable(merge_sort merge_sort.cpp)
#add_executable(producer_consumer producer_consumer.cpp)
# add_executable(producer_consumer_percussion producer_consumer_percussion.cpp)
#add_executable(progress progress.cpp)
add_executable(quick_sort quick_sort.cpp)
# add_executable(rain rain.cpp)
# add_executable(rain_p rain_p.cpp)
add_executable(selection_sort selection_sort.cpp)

# Link libraries
target_link_libraries(bubble_sort tsal)
# target_link_libraries(canon tsal)
# target_link_libraries(chord tsal)
# target_link_libraries(chord_p tsal)
target_link_libraries(hello_tsal tsal)
target_link_libraries(hello_tsal_threads tsal)
# target_link_libraries(jubilate_p tsal)
target_link_libraries(merge_sort tsal)
# target_link_libraries(producer_consumer tsal)
# target_link_libraries(producer_consumer_percussion tsal)
# target_link_libraries(progress tsal)
target_link_libraries(quick_sort tsal)
# target_link_libraries(rain tsal)
# target_link_libraries(rain_p tsal)
target_link_libraries(selection_sort tsal)
#include "tsal.hpp"
#include <omp.h>

/** @example hello_tsal_threads.cpp
 *
```

```
 * A more complex "Hello World!" example that uses multithreading
 * The main idea is that each synth gets its own thread, and then plays
 * a pitch based on that thread id.
 */
int main(int argc, char* argv[]) {
  int numThreads = (argc == 2) ? atoi(argv[1]) : 2;
  if (numThreads == 0) {
    std::cout << "\nhello_tsal_threads <num_threads>\n"
              << "\tnum_threads = the number of threads\n"
              << std::endl;
    return 0;
  }

  // Create the mixer and array of synths
  tsal::Mixer mixer;

  // Setup omp with a number of threads
  omp_set_num_threads(numThreads);

  // Run the parallel block
  #pragma omp parallel
  {
    int id = omp_get_thread_num();
    tsal::Synth synth;
    // Add the synth to the mixer
    mixer.add(synth);
    synth.setMode(tsal::Oscillator::SAW);
    // Start playing a pitch based on thread number
    synth.play(tsal::C4 + 5 * id);
    // Wait for a second
    tsal::Util::thread_sleep(5000);
    // Stop playing
    synth.stop();
  }

  return 0;
}
#include "tsal.hpp"

/** @example hello_tsal.cpp
 *
 * A basic "Hello World!" application for TSAL
 *
```

```
 * Mixer - Basically the engine of TSAL. It initializes all the necessary things
 * to start an audio stream. It contains the master channel which other channels
 * and instruments are routed to.\n
 * Synth - A very basic audio synthesizer. It can only play one note at a time
(monophonic),
 * and it can choose from 3 basic waveforms: sine, saw, and square.
 */
using namespace tsal;

int main() {
  // Create the mixer object
  Mixer mixer;
  // sf.play(C4);
  // sf.play(E4);
  // Create the synthesizer object
  PolySynth synth;
  PolySynth synth2;
  Delay delay;
  delay.setParameter(Delay::FEEDBACK, 0.3);
  // mixer.add(delay);
  // Synth synth;
  // synth.setMode(Oscillator::SAW);
  // synth.setPanning(-0.8);
  synth.setParameter(PolySynth::MODULATION_MODE, Oscillator::AM);
  synth.setParameter(PolySynth::VOLUME, 1.0);
  synth.setParameter(PolySynth::ENV_ATTACK, 0.0);
  synth.setParameter(PolySynth::LFO_ACTIVE, 1.0);
  synth.setParameter(PolySynth::LFO_ATTACK, 3.0);
  synth.setParameter(PolySynth::LFO_MODE, Oscillator::SQUARE);
  synth.setParameter(PolySynth::LFO_FREQUENCY, 2.0);
  // synth.setParameter(PolySynth::PANNING, -1.0);
  // synth.setParameter(PolySynth::OSC2_OFFSET, 0.2);
  synth.setParameter(PolySynth::ENV_ATTACK, 0.1);
  // synth.setParameter(PolySynth::ENV_RELEASE, 10.0);
  synth.setParameter(PolySynth::OSC1_MODE, Oscillator::SINE);
  synth.setParameter(PolySynth::OSC2_MODE, Oscillator::AM);
  synth2.setParameter(PolySynth::OSC1_MODE, Oscillator::SINE);
  // synth2.setParameter(PolySynth::PANNING, 1.0);
  synth.setParameter(PolySynth::MODULATION_MODE, Oscillator::NONE);
  synth2.setParameter(PolySynth::ENV_RELEASE, 3.0);
  // synth.setParameter(PolySynth::OSC2_MODE, Oscillator::WHITE_NOISE);

  synth2.setParameter(PolySynth::VOLUME, 0.5);
```

```
// PolySynth synth3 = synth;
// PolySynth synth(std::move(synth2));
// Add the synth t
Channel chan;
mixer.add(synth2);
chan.setParameter(Channel::PANNING, 1.0);
mixer.add(chan);

synth.play(C4);
synth.setParameter(PolySynth::MODULATION_MODE, Oscillator::MIX);
// Play a note on the synth
synth2.play(E4);
synth2.stop(E4);

// synth.setParameter(PolySynth::OSC1_MODE, Oscillator::SQUARE);
// synth.setParameter(PolySynth::OSC2_OFFSET, 0.0);
// synth.setParameter(PolySynth::OSC2_MODE, Oscillator::SAW);
synth2.setParameter(PolySynth::MODULATION_MODE, Oscillator::PM);

synth2.play(E4);
synth2.stop(E4);
// synth.play(G4);
synth.stop(C4);

for (unsigned i = 0; i < 50; i++) {
  static double volume = 0.0;
  synth2.setParameter(PolySynth::VOLUME, volume);
  synth2.play(C4);
  synth2.play(E4);
  synth2.play(G4);
  Util::thread_sleep(500);
  synth2.stop(C4);
  synth2.stop(E4);
  synth2.stop(G4);
  Util::thread_sleep(500);
  volume += 0.1;
}
// Wait for the user to stop the synth
// char input;
// std::cout << "Press <enter> to quit:" << std::flush;
// std::cin.get(input);
```

```
  return 0;

}
#include "tsal.hpp"

using namespace tsal;

#define MAX_VALUE 100000

void bubbleSort(ThreadSynth& synth, int size, int data[]) {
  int temp;
  for (int i = 0; i < size; i++) {
    for (int j = size - 1; j > i; j--) {

      synth.play(C3 + 40 * ((double) data[j] / MAX_VALUE), Timing::MICROSECOND, 50);

      if (data[j] < data[j - 1]) {
        temp = data[j];
        data[j] = data[j - 1];
        data[j - 1] = temp;
      }
    }
  }
}

int main() {
  Mixer mixer;
  ThreadSynth synth;
  mixer.add(synth);
  synth.setEnvelopeActive(false);

  // Generate the data
  const int size = 5000;
  int data[size];
  for (int i = 0; i < size; i++) {
    data[i] = rand() % MAX_VALUE;
  }
  // Sort the data
  bubbleSort(synth, size, data);
}
#include "tsal.hpp"

using namespace tsal;
```

```
#define MAX_VALUE 10000

void selectionSort(ThreadSynth& synth, int size, int* data) {
  int min;
  int temp;
  for (int i = 0; i < size; i++) {
    min = i;
    for (int j = i; j < size; j++) {
      synth.play(C2 + 55 * ((double) data[j] / MAX_VALUE), Timing::MICROSECOND, 100);

      min = data[j] < data[min] ? j : min;
    }
    temp = data[i];
    data[i] = data[min];
    data[min] = temp;
  }
}

int main() {
  Mixer mixer;
  ThreadSynth synth;
  mixer.add(synth);
  synth.setEnvelopeActive(false);

  // Generate the data
  const int size = 500;
  int* data = new int[size];
  for (int i = 0; i < size; i++) {
    data[i] = rand() % MAX_VALUE;
  }
  // Sort the data
  selectionSort(synth, size, data);
}
#include "tsal.hpp"

using namespace tsal;

#define MAX_VALUE 100000

void insertionSort(ThreadSynth& synth, int size, int* data) {
  int insertValue;
  int j;
```

```
  for (int i = 1; i < size; i++) {
    insertValue = data[i];
    j = i;
    while (j > 0 && data[j - 1] > insertValue) {
      synth.play(C3 + 45 * ((double) data[j] / MAX_VALUE), Timing::MICROSECOND, 50);

      data[j] = data[j - 1];
      j--;
    }
    data[j] = insertValue;
  }
}

int main() {
  Mixer mixer;
  ThreadSynth synth;
  mixer.add(synth);
  synth.setEnvelopeActive(false);

  // Generate data
  const int size = 5000;
  int* data = new int[size];
  for (int i = 0; i < size; i++) {
    data[i] = rand() % MAX_VALUE;
  }
  // Sort the data
  insertionSort(synth, size, data);
}

#include "tsal.hpp"
#include <omp.h>
#include <iostream>

/** @example progress.cpp
 *
 * ProgressOctave is the audio equivalent of a graphical progress bar
 * The progress is conveyed through the ascending pitch. No progress
 * means low pitch, and an almost completed task will be playing at a high
 * pitch. When done in parallel, each thread takes a chunk of the pitch
 * range based on how the omp parallel for pragma divides up the work
 *
 * Parse the parameters\n
 * Create the mixer and ProgressOctave\n
```

```
 * Run parallel block
 * - Update the ProgressOctave
 */
int main(int argc, char* argv[]) {
  const unsigned numThreads = (argc < 2) ? 2 : (unsigned) atoi(argv[1]);
  const unsigned work = (argc < 3) ? 100000000 : (unsigned) atoi(argv[2]);

  tsal::Mixer mixer;
  tsal::ProgressOctave progress(tsal::C4, work, numThreads);

  mixer.add(progress);

  omp_set_num_threads(numThreads);
  #pragma omp parallel for
  for (unsigned i = 0; i < work; i++) {
    progress.update(omp_get_thread_num());
  }

  return 0;
}
#include "tsal.hpp"
#include <thread>
#include <condition_variable>
#include <iostream>
#include <queue>

class SharedQueue {
  public:
    void produce(int item) {
      std::unique_lock<std::mutex> lk(mMutex);
      mCondition.wait(lk, [this]{return mQueue.size() < MaxQueueSize;});
      mQueue.push(item);
      tsal::Util::thread_sleep(rand() % 2000);
      lk.unlock();
      mCondition.notify_all();
    };
    int consume() {
      std::unique_lock<std::mutex> lk(mMutex);
      mCondition.wait(lk, [this]{return mQueue.size() > 0;});
      const int front = mQueue.front();
      mQueue.pop();
      tsal::Util::thread_sleep(rand() % 2000);
      lk.unlock();
```

```
    mCondition.notify_all();
    return front;
  };
  unsigned size() {
    return mQueue.size();
  }
  unsigned MaxQueueSize = 12;
 private:
  std::condition_variable mCondition;
  std::mutex mMutex;
  std::queue<int> mQueue;
};

void monitor(SharedQueue* queue, tsal::Mixer* mixer) {
 tsal::SoundFont sf("/usr/share/soundfonts/FluidR3_GM.sf2");
 sf.setPreset("Halo Pad");
 mixer->add(sf);

 int size = 0;
 sf.play(40);
 while(true) {
  tsal::Util::thread_sleep(1000);
  int currentSize = queue->size();
  if (size != currentSize) {
   sf.stop(40 + size);
   sf.play(40 + currentSize);
   size = currentSize;
  }
 }
}

void produce(SharedQueue* queue, tsal::Mixer* mixer) {
 tsal::SoundFont sf("/usr/share/soundfonts/FluidR3_GM.sf2");
 sf.setPreset(1);
 mixer->add(sf);

 int item;
 while (true) {
  tsal::Util::thread_sleep(rand() % 2000);
  item = rand() % 50;
  std::cout << "producing: " << item << std::endl;
  queue->produce(item);
  sf.play(30 + item);
```

```
    tsal::Util::thread_sleep(500);
    sf.stop(30 + item);

  }
}

void consume(SharedQueue* queue, tsal::Mixer* mixer) {
  tsal::SoundFont sf("/usr/share/soundfonts/FluidR3_GM.sf2");
  sf.setPreset(4);
  mixer->add(sf);

  int item;
  while (true) {
    tsal::Util::thread_sleep(rand() % 2000);
    item = queue->consume();
    std::cout << "consuming: " << item << std::endl;
    sf.play(50 + item);
    tsal::Util::thread_sleep(500);
    sf.stop(50 + item);
  }
}

/** @example producer_consumer.cpp
 *
 * Producer and consumer is a common example of synchonization between multiple
processes or threads.
 * The SharedQueue class acts as a monitor between the producers and consumers. The
sine wave
 * pitch corresponds to the fullness of the queue, the square wave blips correspond to a
production,
 * and the saw wave blips correspond to consumption
 *
 * Parse the parameter\n
 * Create the mixer and the SharedQueue
 * Start the producer and consumer threads
 * Producer threads:
 * - Make a random item
 * - Add the item to the queue
 * - Play a square wave with pitch based on item
 * Consumer threads:
 * - Consume an item from the queue
 * - Play a saw wave with pitch based on item
 * SharedQueue:
```

```
 * - Provide the synchronized access to the underlying queue
 * - Change the pitch of the sine wave whenever a item is added or removed
 */
int main(int argc, char* argv[]) {
  if (argc != 3) {
    std::cout << "Invalid arguments\n\n"
              << "producer_consumer <producers> <consumers>\n"
              << "\tproducers = the number of producers\n"
              << "\tconsumers = the number of consumers\n"
              << std::endl;
    return 0;
  }
  const int numProducers = atoi(argv[1]);
  const int numConsumers = atoi(argv[2]);
  tsal::Mixer mixer;

  SharedQueue queue;

  std::thread producers[numProducers];
  std::thread consumers[numConsumers];
  for (int i = 0; i < numProducers; i++) {
    producers[i] = std::thread(produce, &queue, mixer.getContext());
  }
  for (int i = 0; i < numConsumers; i++) {
    consumers[i] = std::thread(consume, &queue, &mixer);
  }
  std::thread monitor_thread = std::thread(monitor, &queue, &mixer);

  for (int i = 0; i < numProducers; i++) {
    producers[i].join();
  }
  for (int i = 0; i < numConsumers; i++) {
    consumers[i].join();
  }
  monitor_thread.join();

  return 0;
}
#include "tsal.hpp"
#include "MidiFile.h"
#include <omp.h>
#include <vector>
```

```
/** @example rain.cpp
 *
 * One use of multithreaded is to speed up the processing of data. However, in the case of audio,
 * processing a song at twice the speed doesn't really make sense. As a result, this example was created
 * as the next best thing. To make use of the speed up from multithreading, this example parses
 * MIDI files that store the entire song on one track. When processed with a single ThreadSynth,
 * the song will play its parts separately. But when processed with the right number of threads. The
 * ThreadSynths will be playing in unison, and the song will be completed properly.
 *
 * Parse the parameters\n
 * Create the mixer and synths\n
 * Play the song through once with only one thread\n
 * Run parallel block
 * - Calculate a time offset so the later notes get scheduled at the right time
 * - Play the midi events
 */
int main(int argc, char* argv[]) {
  if (argc != 3) {
    std::cout << "Invalid arguments\n\n"
              << "rain <midifile> <tracks>\n"
              << "\tmidifile = a path to a midifile\n"
              << "\ttracks = the number of tracks in the midifile\n"
              << std::endl;
    return 0;
  }
  const int numTracks = atoi(argv[2]);
  tsal::MidiParser midiParser(numTracks, argv[1]);

  tsal::Mixer mixer;
  std::vector<tsal::ThreadSynth> voices(numTracks,
tsal::ThreadSynth(mixer.getContext()));

  omp_set_num_threads(numTracks);

  #pragma omp parallel
  {
    int id = omp_get_thread_num();
    mixer.add(voices[id]);
```

```
    voices[id].setVolume(0.3);
    voices[id].setEnvelope(0, 0, 1, 2.0);

    int timeOffset = midiParser[id *
std::floor(midiParser.size()/omp_get_num_threads())].tick;

    #pragma omp for
    for (unsigned i = 0; i < midiParser.size(); i++) {
      auto& me = midiParser[i];
      if (me.isNoteOn())
        voices[id].play(me.getKeyNumber(), tsal::Sequencer::TICK, me.tick - timeOffset);

      if (me.isNoteOff())
        voices[id].stop(tsal::Sequencer::TICK, me.tick - timeOffset);
    }
  }
  return 0;
}
# Portaudio
add_subdirectory(portaudio EXCLUDE_FROM_ALL)
# TinySoundFont
set(TSF_INCLUDE_DIR ${PROJECT_SOURCE_DIR}/extern/TinySoundFont
  CACHE PATH "TinySoundFont include directory")

set(LADSPA_INCLUDE_DIR ${PROJECT_SOURCE_DIR}/extern/ladspa
  CACHE PATH "ladspa include directory")

# also install it
install(FILES ${LADSPA_INCLUDE_DIR}/ladspa.h DESTINATION
${CMAKE_INSTALL_INCLUDEDIR}/tsal)
install(FILES ${TSF_INCLUDE_DIR}/tsf.h DESTINATION
${CMAKE_INSTALL_INCLUDEDIR}/tsal)

# for convenience setup a target
add_library(ladspa INTERFACE)
add_library(tsf INTERFACE)

target_include_directories(ladspa INTERFACE
  $<BUILD_INTERFACE:${LADSPA_INCLUDE_DIR}>
  $<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>)

target_include_directories(tsf INTERFACE
  $<BUILD_INTERFACE:${TSF_INCLUDE_DIR}>
```

```
  $<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>)

# need to export target as well
install(TARGETS ladspa
  EXPORT ladspaTargets
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  )

install(TARGETS tsf
  EXPORT tsfTargets
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  )
#include "ChannelDevice.hpp"

namespace tsal {
}
#include "Panning.hpp"

namespace tsal {

ParameterManager::Parameter Panning::PanningParameter = { .name="Panning",
.min=-1.0, .max=1.0, .defaultValue=0.0 };

void Panning::setChannelPanning(unsigned channelCount) {
  switch (channelCount) {
    // mono channel
  case 1: {
    mChannelPanning.resize(1);
    mChannelPanning[0] = 1.0 - std::abs(mPanning);
    break;
  }
    // stereo channel
  case 2: {
    const double delta = mPanning * M_PI/4.0;
    mChannelPanning.resize(2);
    const auto s = sin(delta);
    const auto c = cos(delta);
    mChannelPanning[0] = panningConstant * (c + s);
    mChannelPanning[1] = panningConstant * (c - s);
```

```
    break;
  }
  }
};

double Panning::panningConstant = std::sqrt(2.0)/2.0;

}
#include "PolySynth.hpp"
#include "Envelope.hpp"
#include "Oscillator.hpp"

namespace tsal {

std::vector<ParameterManager::Parameter> PolySynth::PolySynthParameters
({
  { .name="Osc1 Mode", .min=0.0, .max=3.0, .defaultValue=0.0 },
  { .name="Osc2 Mode", .min=0.0, .max=3.0, .defaultValue=0.0 },
  { .name="Osc2 Offset", .min=0.0, .max=1.0, .defaultValue=0.0},
  { .name="Modulation Mode", .min=0.0, .max=3.0, .defaultValue=0.0 },
  { .name="Attack", .min=0.0, .max=100.0, .defaultValue=0.0, .exclusiveMin=true },
  { .name="Decay", .min=0.0, .max=100.0, .defaultValue=0.5, .exclusiveMin=true },
  { .name="Sustain", .min=0.0, .max=1.0, .defaultValue=0.5, .exclusiveMin=true },
  { .name="Release", .min=0.0, .max=100.0, .defaultValue=2.0, .exclusiveMin=true },
  { .name="LFO Active", .min=0.0, .max=1.0, .defaultValue=0.0 },
  { .name="LFO Mode", .min=0.0, .max=3.0, .defaultValue=0.0 },
  { .name="LFO Frequency", .min=0.0, .max=15000.0, .defaultValue=1.0 },
  { .name="LFO Attack", .min=0.0, .max=100.0, .defaultValue=0.0, .exclusiveMin=true },
  { .name="LFO Decay", .min=0.0, .max=100.0, .defaultValue=0.5, .exclusiveMin=true },
  { .name="LFO Sustain", .min=0.0, .max=1.0, .defaultValue=0.5, .exclusiveMin=true },
  { .name="LFO Release", .min=0.0, .max=100.0, .defaultValue=2.0, .exclusiveMin=true },
});

void PolySynth::getOutput(AudioBuffer<float> &buffer) {
  if (!mActive) {
    return;
  }
  const auto channels = buffer.getChannelCount();
  const auto frames = buffer.getFrameCount();

  mPanning.setChannelPanning(channels);

  for (unsigned long i = 0; i < frames; i++) {
```

```
    // Get the collective output of the voices
    double lfo = ((getParameter(LFO_ACTIVE) > 0.5 ? mLFO.getSample() : 1.0) + 1) / 2;
    double output = 0.0;
    double activeVoices = 0.0;
    for (size_t k = 0; k < mVoices.size(); k++) {
      if (mVoices[k].isActive()) {
        output += mVoices[k].getSample(lfo);
        ++activeVoices;
      }
    }
    // output += mVoices[0].getSample();
    // Scale the output by the number of active voices
    output = activeVoices > 0 ? output / activeVoices : output;

    for (unsigned j = 0; j < channels; j++) {
      buffer[i * channels + j] = output * mAmp.getAmp() * mPanning.getPanningChannel(j) *
lfo;
    }
  }
}

void PolySynth::updateContext(const Context& context) {
  OutputDevice::updateContext(context);
  for (auto& voice : mVoices) {
    voice.updateContext(context);
  }
  mLFO.updateContext(context);
}

/* @brief Play a note with velocity
 *
 * @param note
 * @param velocity
 */
void PolySynth::play(double note, double velocity) {
  Voice* voice = getInactiveVoice();
  if (voice == nullptr) {
    voice = &mVoices[0];
    voice->stop();
  }
  voice->play(note, velocity);
}
```

```
/* @brief Stop playing a note
 *
 * @param note
 */
void PolySynth::stop(double note) {
  for (auto& voice : mVoices) {
    if (voice.getNote() == note) {
      voice.stop();
      voice.setActive(false);
    }
  }
}

/* @brief Set the mode of the underlying synths
 *
 * @param mode
 */
void PolySynth::setMode(Oscillator::OscillatorMode mode) {
  for (unsigned i = 0; i < mVoices.size(); i++) {
  }
}

PolySynth::Voice* PolySynth::getInactiveVoice() {
  // Whenever a note is pressed, an inactive voice needs to be found an played
  // If all the voices are active, a nullptr is returned
  Voice* voice = nullptr;
  for (unsigned i = 0; i < mVoices.size(); i++) {
    if (!mVoices[i].isActive()) {
      mVoices[i].setActive();
      voice = &mVoices[i];
      break;
    }
  }
  return voice;
}

double PolySynth::Voice::getSample(double lfo) {
  mOsc1.setParameter(Oscillator::MODULATION, mOsc2.getSample());
  return lfo * mLFOEnvelope.getSample() * mFilter.process(mOsc1.getSample() *
                    (mVelocity / 120.0) *
                    mEnvelope.getSample());
}
```

```
void PolySynth::Voice::play(double note, double velocity) {
  mNote = note;
  setActive();
  mOsc1.setActive();
  mOsc2.setActive();
  mEnvelope.start();
  mLFOEnvelope.start();
  mVelocity = velocity;
  mOsc1.setNote(note);
  mOsc2.setNote(note);
}

void PolySynth::Voice::stop(double note) {
  (void)note;
  if (mEnvelope.isActive()) {
    mEnvelope.stop();
    mLFOEnvelope.stop();
  } else {
    mOsc1.setActive(false);
    mOsc2.setActive(false);
  }
}

void PolySynth::Voice::updateContext(const Context& context) {
    mOsc1.updateContext(context);
    mOsc2.updateContext(context);
    mEnvelope.updateContext(context);
    mLFOEnvelope.updateContext(context);
}

}
#include "LadspaManager.hpp"

namespace tsal {

std::vector<ladspa_key> LadspaManager::listPlugins() {
  // std::vector<std::string> ladspaDirectories;
  // std::istringstream ladspaEnv(getenv("LADSPA_PATH"));
  // std::string s;
  // while (std::getline(ladspaEnv, s, ':')) {
  //   std::cout << s << std::endl;
  //   ladspaDirectories.push_back(s);
  // }
```

```
  std::vector<ladspa_key> ladspa_keys;
  return ladspa_keys;
}

const LADSPA_Descriptor * LadspaManager::loadPlugin(const std::string& pluginPath) {
  const LADSPA_Descriptor * descriptor = NULL;
  /*
  void * pluginHandle = dlopen(pluginPath.c_str(), RTLD_NOW);
  if (pluginHandle == NULL) {
    std::cout << "Failed to open plugin: " << pluginPath << std::endl;

  }

  LADSPA_Descriptor_Function descriptorFunction =
(LADSPA_Descriptor_Function)dlsym(pluginHandle, "ladspa_descriptor");

  for (long pluginIndex = 0;; pluginIndex++) {
    descriptor = descriptorFunction(pluginIndex);
    if (descriptor == NULL) {
      std::cout << "Failed to initialize plugin " << pluginPath << std::endl;
      return NULL;
    }
    return descriptor;
  }
  */
  return descriptor;
}

}
#include "ProgressOctave.hpp"

namespace tsal {

ProgressOctave::ProgressOctave(unsigned startNote, unsigned problemSize, unsigned
numWorkers) {
  mStartNote = startNote;
  mProblemSize = problemSize;
  mNumWorkers = numWorkers;
}

void ProgressOctave::getOutput(AudioBuffer<float> &buffer) {
  mRoutedOscillators.getOutput(buffer);
}
```

```
void ProgressOctave::updateContext(const Context& context) {
  OutputDevice::updateContext(context);
  mRoutedOscillators.updateContext(context);

  double startingFrequency = Oscillator::getFrequencyFromNote(mStartNote);
  // Jump by 2 octaves, basically 3 * startingFrequency
  double octavePortion = (3 * startingFrequency) / mNumWorkers;
  mStepValue = octavePortion / mProblemSize;
  for (unsigned i = 0; i < mNumWorkers; i++) {
    mOscillators.push_back(std::make_unique<Oscillator>());

    auto& oscillator = mOscillators[i];
    mRoutedOscillators.add(*(oscillator.get()));
    oscillator->setFrequency(startingFrequency + octavePortion * i);
  }
}

/* @brief Update the oscillator that corresponds to the id
 *
 * @param id
 */
void ProgressOctave::update(unsigned id) {
  auto& oscillator = mOscillators[id];
  oscillator->setFrequency(oscillator->getFrequency() + mStepValue);
}

}
#include "Channel.hpp"

namespace tsal {

std::vector<ParameterManager::Parameter> Channel::ChannelParameters
({
  { .name="Volume", .min=0.0, .max=2.0, .defaultValue=1.0 },
  { .name="Panning", .min=-1.0, .max=1.0, .defaultValue=0.0 },
});

Channel::Channel() :
  ParameterManager(ChannelParameters) {
  mChannelIn.add(mRoutedInstruments);
  mChannelIn.add(mRoutedChannels);
}
```

```
Channel::~Channel() {
  if (mParentChannel != nullptr)
    mParentChannel->remove(*this);
}

void Channel::setParentChannel(Channel* channel) {
  if (mParentChannel != nullptr)
    mParentChannel->remove(*this);
  mParentChannel = channel;
}

void Channel::updateContext(const Context& context) {
  OutputDevice::updateContext(context);
  mChannelIn.updateContext(context);
  mEffectChain.updateContext(context);
}

void Channel::getOutput(AudioBuffer<float> &buffer) {
  if (mActive) {
    const auto channels = buffer.getChannelCount();
    const auto frames = buffer.getFrameCount();

    mPanning.setChannelPanning(channels);

    mChannelIn.getOutput(buffer);
    mEffectChain.getOutput(buffer);
    for (unsigned i = 0; i < frames; i++) {
      for (unsigned j = 0; j < channels; j++) {
        buffer[i * channels + j] *= mAmp.getAmp() * mPanning.getPanningChannel(j);
      }
    }
  }
}

/**
 * @brief Add an effect to the end of the effect chain
 *
 * @param effect
 */
void Channel::add(Effect& effect) {
  effect.setParentChannel(this);
  mEffectChain.add(effect);
```

```
}
/**
 * @brief Remove an effect from the effect chain
 *
 * @param effect
 */
void Channel::remove(Effect& effect) {
  mEffectChain.remove(effect);
}

/**
 * @brief Add an instrument to the channel
 *
* @param instrument
 */
void Channel::add(Instrument& instrument) {
  instrument.setParentChannel(this);
  mRoutedInstruments.add(instrument);
}

/**
 * @brief Remove an instrument from the channel
 *
 * @param instrument
 */
void Channel::remove(Instrument& instrument) {
  mRoutedInstruments.remove(instrument);
}

/**
 * @brief Route another channel through the current one
 *
 * @param channel
 */
void Channel::add(Channel& channel) {
  if (&channel != this) {
    channel.setParentChannel(this);
    mRoutedChannels.add(channel);
  }
}

/**
 * @brief Remove a routed channel
```

```
 *
 * @param channel
 */
void Channel::remove(Channel& channel) {
  mRoutedChannels.remove(channel);
}

void Channel::parameterUpdate(unsigned id) {
  switch(id) {
  case VOLUME:
    mAmp.setVolume(getParameter(VOLUME));
    break;
  case PANNING:
    mPanning.setPanning(getParameter(PANNING));
    break;
  }
};

}
#include "Amp.hpp"

namespace tsal {

ParameterManager::Parameter Amp::AmpParameterVolume = { .name="Volume",
.min=0.0, .max=2.0, .defaultValue=1.0 };

}
#include "Util.hpp"
#include <random>

namespace tsal {

void Util::thread_sleep(unsigned duration, const Timing::TimeScale scale) {
  std::this_thread::sleep_for(duration * std::chrono::microseconds(scale));
}

double Util::ampToDb(double amplitude) {
  return 20.0 * std::log(amplitude) / M_LN10;
}

double Util::dbToAmp(double db) {
  return std::pow(10.0, db / 20.0);
}
```

```
double Util::volumeToDb(double volume) {
  return 33.22 * std::log10(volume);
}

double Util::dbToVolume(double db) {
  return std::pow(10.0, db / 33.22);
}


std::random_device rd;
std::mt19937 Util::mGen(rd());
std::uniform_real_distribution<> Util::mDist(0.0, 1.0);

}
#include "Instrument.hpp"
#include "Channel.hpp"

namespace tsal {

std::vector<ParameterManager::Parameter> Instrument::InstrumentParameters
({
  { .name="Volume", .min=0.0, .max=2.0, .defaultValue=1.0 },
  { .name="Panning", .min=-1.0, .max=1.0, .defaultValue=0.0 },
});

Instrument::~Instrument() {
  if (mParentChannel != nullptr)
    mParentChannel->remove(*this);
}

void Instrument::setParentChannel(Channel* channel) {
  if (mParentChannel != nullptr)
    mParentChannel->remove(*this);
  mParentChannel = channel;
}

void Instrument::parameterUpdate(unsigned id) {
  switch(id) {
  case VOLUME:
    mAmp.setVolume(getParameter(VOLUME));
    break;
  case PANNING:
```

```
    mPanning.setPanning(getParameter(PANNING));
    break;
  }
}

}

#include "OutputDevice.hpp"
#include "Util.hpp"
#include <cmath>

namespace tsal {

/**
 * @brief Set the active status of the device
 *
 * It is up to the class how to implement the active status
 *
 * @param active o
 */
void OutputDevice::setActive(bool active) {
  mActive = active;
}

/**
 * @brief Check if the device is active
 *
 * @return bool
 */
bool OutputDevice::isActive() const {
  return mActive;
}

// Util::ParameterRange<double> OutputDevice::mGainRange = std::make_pair(-50.0,
50.0);
// Util::ParameterRange<double> OutputDevice::mVolumeRange = std::make_pair(0.0,
2.0);

}
#include "Compressor.hpp"
#include "Util.hpp"
#include <iostream>
```

```
namespace tsal {

std::vector<ParameterManager::Parameter> Compressor::CompressorParameters
({
  { .name="Release", .min=0.0, .max=2000.0, .defaultValue=0.0 },
  { .name="Threshold", .min=-60.0, .max=60.0, .defaultValue=-30.0 },
  { .name="Ratio", .min=1.0, .max=20.0, .defaultValue=2.0 },
  { .name="Pre Gain", .min=-30.0, .max=30.0, .defaultValue=0.0 },
  { .name="Post Gain", .min=-30.0, .max=30.0, .defaultValue=0.0 },
});

void Compressor::getOutput(AudioBuffer<float> &buffer) {
  /* The Compressor uses a circular buffer where a value is written behind
   * the value that was read. Once the buffer is full of new values, all
   * of the samples in the buffer are processed.
   */
  // If not active, just route the samples through without applying an filtering
  if (!mActive) {
    return;
  }

  filterAudio(buffer);

  // Get an audio sample from the audio that has been processed in front
  // return mBuffer[mCurrentSample];
}

void Compressor::updateContext(const Context& context) {
  OutputDevice::updateContext(context);
  parameterUpdate(ATTACK);
  parameterUpdate(RELEASE);
}

/**
 * @brief Get the sound envelope for the sample buffer
 *
 */
void Compressor::getEnvelope(AudioBuffer<float> &buffer) {
  const auto channels = buffer.getChannelCount();
  const auto frames = buffer.getFrameCount();
  mEnvelope.resize(frames);
  for (unsigned i = 0; i < channels; i++) {
    for (unsigned long j = 0; j < frames; j++) {
```

```
      float envIn = std::abs(buffer[j * channels + i]);
      double gain = mEnvelopeSample < envIn ? mAttackGain : mReleaseGain;
      mEnvelopeSample = envIn + gain * (mEnvelopeSample - envIn);
      if (i == 0) {
        mEnvelope[j] = mEnvelopeSample / (float) channels;
      } else {
        mEnvelope[j] += mEnvelopeSample / (float) channels;
      }
    }
  }
}
/**
 * @brief Compress the audio in the buffer if necessary
 *
 */
void Compressor::filterAudio(AudioBuffer<float> &buffer) {
  double postGainAmp = Util::dbToAmp(getParameter(POST_GAIN));

  // If there is any pregain, apply it to the audio buffer
  if (getParameter(PRE_GAIN) != 0.0) {
    double preGainAmp = Util::dbToAmp(getParameter(PRE_GAIN));
    for (unsigned i = 0; i < buffer.size(); i++) {
      buffer[i] *= preGainAmp;
    }
  }

  getEnvelope(buffer);
  calculateSlope();

  const auto channels = buffer.getChannelCount();
  const auto frames = buffer.getFrameCount();
  // Apply the adjusted gain and postGain to the audio buffer
  for (unsigned long i = 0; i < frames; i++) {
    mGain = mSlope * (getParameter(THRESHOLD) - Util::ampToDb(mEnvelope[i]));
    mGain = std::min(0.0, mGain);
    mGain = Util::dbToAmp(mGain);
    for (unsigned j = 0; j < channels; j++) {
      buffer[i * channels + j] *= (mGain * postGainAmp);
    }
  }
}

/**
```

```
 *
 * @brief Get the slope based off the ratio
 *
 */
void Compressor::calculateSlope() {
  mSlope = 1.0 - (1.0 / getParameter(RATIO));
}

/**
 * @brief Set the attack time (ms)
 *
 * @param attackTime
 */
void Compressor::setAttackTime(double attackTime) {
  setParameter(ATTACK, attackTime);
}

/**
 * @brief Set the release time (ms)
 *
 * @param releaseTime
 */
void Compressor::setReleaseTime(double releaseTime) {
  setParameter(RELEASE, releaseTime);
}

/**
 * @brief Set the threshold
 *
 * @param threshold (dB)
 */
void Compressor::setThreshold(double threshold) {
  setParameter(THRESHOLD, threshold);
}

/**
 * @brief Set the ratio
 *
 * @param ratio (1: n)
 */
void Compressor::setRatio(double ratio) {
  setParameter(RATIO, ratio);
}
```

```
/**
 * @brief Set the pre gain
 *
 * @param preGain (dB)
 */
void Compressor::setPreGain(double preGain) {
  setParameter(PRE_GAIN, preGain);
}

/**
 * @brief Set the post gain
 *
 * @param postGain (dB)
 */
void Compressor::setPostGain(double postGain) {
  setParameter(POST_GAIN, postGain);
}

void Compressor::parameterUpdate(unsigned id) {
  switch(id) {
  case ATTACK:
    mAttackGain = getParameter(id) == 0.0 ? 0.0 : std::exp(-1.0 /
(mContext.getSampleRate() * getParameter(id)/1000));
    break;
  case RELEASE:
    mReleaseGain = getParameter(id) == 0.0 ? 0.0 : std::exp(-1.0 /
(mContext.getSampleRate() * getParameter(id)/1000));
    break;
  }
}

}
# file(GLOB HEADER_LIST CONFIGURE_DEPENDS
"${TSAL_SOURCE_DIR}/include/*.hpp")
set(HEADER_LIST "${PROJECT_SOURCE_DIR}/include/tsal.hpp")

set(TSAL_INCLUDE_DIR ${PROJECT_SOURCE_DIR}/include)
set(INSTALL_CONFIGDIR ${CMAKE_INSTALL_LIBDIR}/cmake/tsal)

SET(BUILD_SHARED_LIBS ON)

set(PUBLIC_HEADERS
```

```
  ${TSAL_INCLUDE_DIR}/Amp.hpp
  ${TSAL_INCLUDE_DIR}/AudioBuffer.hpp
  ${TSAL_INCLUDE_DIR}/ChannelDevice.hpp
  ${TSAL_INCLUDE_DIR}/Channel.hpp
  ${TSAL_INCLUDE_DIR}/Compressor.hpp
  ${TSAL_INCLUDE_DIR}/Context.hpp
  ${TSAL_INCLUDE_DIR}/Delay.hpp
  ${TSAL_INCLUDE_DIR}/EffectChain.hpp
  ${TSAL_INCLUDE_DIR}/Effect.hpp
  ${TSAL_INCLUDE_DIR}/Envelope.hpp
  ${TSAL_INCLUDE_DIR}/Filter.hpp
  # ${TSAL_INCLUDE_DIR}/LadspaEffect.hpp
  # ${TSAL_INCLUDE_DIR}/LadspaManager.hpp
  ${TSAL_INCLUDE_DIR}/InputDevice.hpp
  ${TSAL_INCLUDE_DIR}/Instrument.hpp
  ${TSAL_INCLUDE_DIR}/MidiNotes.hpp
  # ${TSAL_INCLUDE_DIR}/MidiParser.hpp
  ${TSAL_INCLUDE_DIR}/Mixer.hpp
  ${TSAL_INCLUDE_DIR}/Oscillator.hpp
  ${TSAL_INCLUDE_DIR}/OutputDevice.hpp
  ${TSAL_INCLUDE_DIR}/Panning.hpp
  ${TSAL_INCLUDE_DIR}/ParameterManager.hpp
  ${TSAL_INCLUDE_DIR}/PolySynth.hpp
  ${TSAL_INCLUDE_DIR}/ProgressOctave.hpp
  ${TSAL_INCLUDE_DIR}/RouteDevice.hpp
  ${TSAL_INCLUDE_DIR}/Sequencer.hpp
  ${TSAL_INCLUDE_DIR}/SoundFont.hpp
  ${TSAL_INCLUDE_DIR}/Synth.hpp
  ${TSAL_INCLUDE_DIR}/ThreadSynth.hpp
  ${TSAL_INCLUDE_DIR}/Timing.hpp
  ${TSAL_INCLUDE_DIR}/tsal.hpp
  ${TSAL_INCLUDE_DIR}/Util.hpp
  )

set(SOURCES
  Amp.cpp
  Channel.cpp
  ChannelDevice.cpp
  Compressor.cpp
  Context.cpp
  Delay.cpp
  EffectChain.cpp
  Effect.cpp
```

```
  Envelope.cpp
  Filter.cpp
  LadspaEffect.cpp
  LadspaManager.cpp
  Instrument.cpp
  # MidiParser.cpp
  Mixer.cpp
  Oscillator.cpp
  OutputDevice.cpp
  Panning.cpp
  PolySynth.cpp
  ProgressOctave.cpp
  Sequencer.cpp
  Synth.cpp
  ThreadSynth.cpp
  Util.cpp
  )

# Make an automatic library - will be static or dynamic based on user setting
add_library(tsal STATIC ${SOURCES} ${HEADER_LIST})
# Also make it accessible via namespace
add_library(${LOCAL_PROJECT_NAMESPACE}::${LOCAL_PROJECT_NAME} ALIAS tsal)

# We need this directory, and users of our library will need it too
target_include_directories(tsal
  PUBLIC
  $<BUILD_INTERFACE:${PROJECT_SOURCE_DIR}/include>
  )

target_link_libraries(tsal
  PUBLIC
    ladspa
    portaudio
    tsf
    OpenMP::OpenMP_CXX
    Threads::Threads
    # dl
  )

if(CMAKE_CXX_COMPILER_ID MATCHES "Clang|AppleClang|GNU")
   # target_compile_options(tsal PRIVATE -Wall -Werror -Wextra -Wunreachable-code
-Wpedantic)
```

```
   # target_compile_options(tsal PRIVATE -Wall -Wextra -Wunreachable-code
-Wpedantic)
   target_compile_options(tsal PRIVATE -Wall)
endif()
if(CMAKE_CXX_COMPILER_ID MATCHES "Clang")
   target_compile_options(tsal PRIVATE -Wweak-vtables -Wexit-time-destructors
-Wglobal-constructors -Wmissing-noreturn )
endif()
if(CMAKE_CXX_COMPILER_ID MATCHES "MSVC")
   target_compile_options(tsal PRIVATE /W4 /w44265 /w44061 /w44062)
endif()

set_target_properties(tsal
  PROPERTIES
  OUTPUT_NAME "tsal"
  )

# IDEs should put the headers in a nice place
# source_group(TREE "${PROJECT_SOURCE_DIR}/include" PREFIX "Header Files"
FILES ${HEADER_LIST})

# install(TARGETS tsal midifile tsf
install(TARGETS tsal ladspa tsf
  EXPORT tsalTargets
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  )


# installation - build tree specific package config files

install(EXPORT tsalTargets
   FILE TsalTargets.cmake
   NAMESPACE ${LOCAL_PROJECT_NAMESPACE}::
   DESTINATION ${INSTALL_CONFIGDIR}
)

install(FILES ${PUBLIC_HEADERS}
   DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}/tsal
   # FILES_MATCHING PATTERN "*.h*"
)
```

```
####################
# ConfigVersion file
##
include(CMakePackageConfigHelpers)
write_basic_package_version_file(
   ${CMAKE_CURRENT_BINARY_DIR}/TsalConfigVersion.cmake
   VERSION ${PROJECT_VERSION}
   COMPATIBILITY AnyNewerVersion
)

configure_package_config_file(
   ${PROJECT_SOURCE_DIR}/cmake/TsalConfig.cmake.in
   ${CMAKE_CURRENT_BINARY_DIR}/TsalConfig.cmake
   INSTALL_DESTINATION ${INSTALL_CONFIGDIR}
)

## Install all the helper files
install(
   FILES
     ${CMAKE_CURRENT_BINARY_DIR}/TsalConfig.cmake
     ${CMAKE_CURRENT_BINARY_DIR}/TsalConfigVersion.cmake
   DESTINATION ${INSTALL_CONFIGDIR}
)
#include "ThreadSynth.hpp"
#include "Sequencer.hpp"

namespace tsal {

void ThreadSynth::play(double note, Timing::TimeScale scale, unsigned duration) {
  Synth::play(note);
  Util::thread_sleep(duration, scale);
}

void ThreadSynth::stop(Timing::TimeScale scale, unsigned duration) {
  Synth::stop();
  Util::thread_sleep(duration, scale);
}

}
#include "Sequencer.hpp"

namespace tsal {
```

```
void Sequencer::tick() {
  // This has some margin of error since mSamplesPerTick is a floating point number
  // But it seems to work well enough, maybe in the future it would be better to change to
a more consistent timing method
  if (++mSampleTime > mSamplesPerTick) {
    mTick++;
    // Keep checking the front until there are no events left for this tick value
    while (!mEventQueue.empty() && mTick >= mEventQueue.top()->tick) {
      // Run the callback function for the event
      mEventQueue.top()->callback();
      // Free the memory
      //delete mEventQueue.top();
      mEventQueue.pop();
    }
    mSampleTime = 0;
  }
}

void Sequencer::setTick(unsigned tick) {
  mTick = tick;
}

/**
 * @brief Set the BPM (beats pre minute)
 *
 * @param bpm
 */
void Sequencer::setBPM(unsigned bpm) {
  mBPM = Util::checkParameterRange("Sequencer: BPM", bpm, mBPMRange);
  setSamplesPerTick();
}

/**
 * @brief Set the PPQ (pulses per quater)
 *
 * @param ppq
 */
void Sequencer::setPPQ(unsigned ppq) {
  mPPQ = Util::checkParameterRange("Sequencer: PPQ", ppq, mPPQRange);
  setSamplesPerTick();
}

void Sequencer::setSamplesPerTick() {
```

```
  mSamplesPerTick = mSampleRate / ((mBPM * mPPQ) / 60);
}

void Sequencer::schedule(std::function<void ()> callback, Timing::NoteScale scale,
unsigned duration) {
  int tick = getTicksInNote(scale) * duration;
  mEventQueue.push(std::make_unique<Event>(mTick + tick, callback));
}

unsigned Sequencer::getTick() const {
  return mTick;
}

unsigned Sequencer::getTicksInNote(Timing::NoteScale note) const {
  return (note == Timing::TICK) ? 1 : mPPQ * 4.0 * (1.0/note);
}

Util::ParameterRange<unsigned> Sequencer::mBPMRange = std::make_pair(1, 1000);
Util::ParameterRange<unsigned> Sequencer::mPPQRange = std::make_pair(1, 1000);

}
#include "LadspaEffect.hpp"
#include "LadspaManager.hpp"
#include <cmath>
#include <ladspa.h>

namespace tsal {

void LadspaEffect::loadPlugin(const std::string& pluginPath) {
  mDescriptor = LadspaManager::loadPlugin(pluginPath);

  mHandle = mDescriptor->instantiate(mDescriptor, mContext.getSampleRate());
  for (unsigned i = 0; i < mDescriptor->PortCount; i++) {
    auto portDescriptor = mDescriptor->PortDescriptors[i];
    if (LADSPA_IS_PORT_CONTROL(portDescriptor)) {
      LADSPA_PortRangeHintDescriptor hintDescriptor =
mDescriptor->PortRangeHints[i].HintDescriptor;
      mControlPorts.push_back(0);
      mDescriptor->connect_port(mHandle, i, &mControlPorts[mControlPorts.size() - 1]);
      std::cout << "Port: " << mDescriptor->PortNames[i] << std::endl;
      if (LADSPA_IS_HINT_SAMPLE_RATE(hintDescriptor)) {
        std::cout << "\tSampleRate Dependent" << std::endl;
      }
```

```
    if (LADSPA_IS_HINT_BOUNDED_BELOW(hintDescriptor)) {
      std::cout << "\tBelow: " << mDescriptor->PortRangeHints[i].LowerBound <<
std::endl;
    }
    if (LADSPA_IS_HINT_BOUNDED_ABOVE(hintDescriptor)) {
      std::cout << "\tUpper: " << mDescriptor->PortRangeHints[i].UpperBound <<
std::endl;
    }
   }
   if (LADSPA_IS_PORT_AUDIO(portDescriptor)) {
    if (LADSPA_IS_PORT_INPUT(portDescriptor)) {
     mInputPorts.push_back(LadspaManager::Port(i));
    }
    if (LADSPA_IS_PORT_OUTPUT(portDescriptor)) {
     mOutputPorts.push_back(LadspaManager::Port(i));
    }
   }
  }

  if (mDescriptor->activate != NULL) {
   mDescriptor->activate(mHandle);
  }
}

void LadspaEffect::getOutput(AudioBuffer<float>& buffer) {
  std::vector<AudioBuffer<LADSPA_Data>*> inputBuffers;
  std::vector<AudioBuffer<LADSPA_Data>*> outputBuffers;
  for (auto& port : mInputPorts) {
   inputBuffers.push_back(&(port.buffer));
  }
  buffer.deinterleaveBuffer(inputBuffers);
  for (auto& port : mInputPorts) {
   mDescriptor->connect_port(mHandle, port.id, port.buffer.getRawPointer());
  }
  for (auto& port : mOutputPorts) {
   port.buffer.setSize(buffer.getFrameCount(), 1);
   mDescriptor->connect_port(mHandle, port.id, port.buffer.getRawPointer());
  }

  mDescriptor->run(mHandle, buffer.getFrameCount());

  for (auto& port : mOutputPorts) {
   outputBuffers.push_back(&(port.buffer));
```

```
  }
  buffer.interleaveBuffers(outputBuffers);
}

}
#include "SoundFont.hpp"

namespace tsal {

SoundFont::SoundFont(std::string filename) {
  // mSoundFont = tsf_load_filename(filename.c_str());
  if (mSoundFont == nullptr) {
    std::cout << "Failed to load SoundFont: " << filename << std::endl;
    return;
  }
  mPresetRange = std::make_pair(0, getPresetCount());
}

SoundFont::~SoundFont() {
  // tsf_close(mSoundFont);
}

// void SoundFont::setMixer(Mixer* mixer) {
//   OutputDevice::updateContext(mContext)
//   tsf_set_output(mSoundFont, TSF_MONO, mContext.getSampleRate(), 0);
// }

void SoundFont::getOutput(AudioBuffer<float> &buffer) {
  // tsf_render_float(mSoundFont, buffer, buffer.size(), 0);
}

void SoundFont::play(double note, double velocity) {
  // tsf_note_on(mSoundFont, mPresetIndex, note, velocity/127.0);
}

void SoundFont::stop(double note) {
  // tsf_note_off(mSoundFont, mPresetIndex, note);
}

void SoundFont::reset() {
  // tsf_reset(mSoundFont);
}
```

```
/**
 * @brief Set the preset (voice) from a preset index
 *
 * @param presetIndex
 */
void SoundFont::setPreset(int presetIndex) {
  mPresetIndex = Util::checkParameterRange("SoundFont: setPreset", presetIndex,
mPresetRange);
}

/**
 * @brief Set the preset (voice) from a preset name
 *
 * @param presetName
 */
void SoundFont::setPreset(std::string presetName) {
  setPreset(getPresetIndex(presetName));
}

/**
 * @brief Get the preset index from a bank and preset number
 * See General MIDI standards
 *
 * @param bank
 * @param presetNumber
 */
int SoundFont::getPresetIndex(int bank, int presetNumber) {
  return 0;//tsf_get_presetindex(mSoundFont, bank, presetNumber);
}

/**
 * @brief Get the preset index from a preset name
 *
 * @param bank
 * @param presetNumber
 */
int SoundFont::getPresetIndex(std::string presetName) {
  for (int i = 0; i < getPresetCount(); i++) {
    if (presetName == getPresetName(i)) {
      return i;
    }
  }
  return -1;
```

```
}

/**
 * @brief Get the number of presets in the soundfont
 */
int SoundFont::getPresetCount() {
  return 0; //tsf_get_presetcount(mSoundFont);
}

/**
 * @brief Get the preset name from a preset index
 *
 * @param presetIndex
 */
std::string SoundFont::getPresetName(int presetIndex) {
  return ""; //std::string(tsf_get_presetname(mSoundFont, presetIndex));
}

/**
 * @brief Get the preset name from a bank and preset number
 * See General MIDI standards
 *
 * @param bank
 * @param presetNumber
 */
std::string SoundFont::getPresetName(int bank, int presetNumber) {
  return ""; //std::string(tsf_bank_get_presetname(mSoundFont, bank, presetNumber));
}

} // namespace tsal
#include "Context.hpp"
#include "Mixer.hpp"

namespace tsal {

Context::Context() :
  mSampleRate(1),
  mChannelCount(1),
  mMixer(nullptr){};

Context::Context(unsigned sampleRate, unsigned channelCount, Mixer *mixer) :
  mSampleRate(sampleRate),
  mChannelCount(channelCount),
```

```
  mMixer(mixer){};

/**
 * @brief Used to add a new device to the model
 *
 * @param change the atomic change to be made
 *
 * Given a function, Context will submit a change request to the mixer which will be
executed after the current audio callback.
 */
void Context::requestModelChange(std::function<void()> change) {
  if (mMixer == nullptr) {
    // This change will not be thread safe
    change();
    return;
  }
  mMixer->requestModelChange(change);
}

}
#include "Envelope.hpp"

namespace tsal {

std::vector<ParameterManager::Parameter> Envelope::EnvelopeParameters
({
  { .name="Attack", .min=0.0, .max=100.0, .defaultValue=0.0, .exclusiveMin=true },
  { .name="Decay", .min=0.0, .max=100.0, .defaultValue=0.5, .exclusiveMin=true },
  { .name="Sustain", .min=0.0, .max=1.0, .defaultValue=0.5, .exclusiveMin=true },
  { .name="Release", .min=0.0, .max=100.0, .defaultValue=2.0, .exclusiveMin=true },
});
/**
 * @brief Increments the current state
 *
 * Each state has its own set of unique properties. Those properties are updated here
 */
void Envelope::updateState() {
  mState = static_cast<EnvelopeState>((mState + 1) % 5);
  mCurrentStateIndex = 0;
  // OFF and SUSTAIN are indefinite so no need to set mNextStateIndex
  mNextStateIndex = stateIsTimed() ? getStateValue(mState) * mContext.getSampleRate()
: 0;
  switch(mState) {
```

```
    case E_OFF:
      mEnvelopeValue = 0.0;
      break;
    case E_ATTACK:
      mEnvelopeValue = getStateValue(E_OFF);
      calculateMultiplier(mEnvelopeValue, 1.0, mNextStateIndex);
      break;
    case E_DECAY:
      mEnvelopeValue = 1.0;
      calculateMultiplier(mEnvelopeValue, getStateValue(E_SUSTAIN), mNextStateIndex);
      break;
    case E_SUSTAIN:
      mEnvelopeValue = getStateValue(E_SUSTAIN);
      break;
    case E_RELEASE:
      calculateMultiplier(mEnvelopeValue, getStateValue(E_OFF), mNextStateIndex);
      break;
  }
}

double Envelope::getStateValue(EnvelopeState state) {
  // This assumes that parameters are ordered
  if (state == E_OFF) {
    return 0.0001;
  } else {
    return getParameter(state);
  }
}

/**
 * @brief Gets the current sample of the envelope
 *
 * @return double A sample of the envelope
 */
double Envelope::getSample() {
  if (!mActive) {
    return 1.0;
  }
  if (stateIsTimed()) {
    if (mCurrentStateIndex >= mNextStateIndex) {
      updateState();
    }
    mCurrentStateIndex++;
```

```
    mEnvelopeValue *= mMultiplier;
  }
  // TODO: never figured out why, but occasionally the envelope Value skyrockets when in
attack state. This is protect against anything ridiculous
  return std::min(1.0, mEnvelopeValue);
}

/**
 * @brief Calculate the multiplier that will be used to update the envelope value
 *
 * Human sense are logarithmic, the envelope needs to change exponentially
 * This function calculates an appropriate multiplier that gets the envelope value
 * from a start level to an end level in the number of specified samples (time)
 *
 * @param startLevel
 * @param endLevel
 * @param lengthInSamples
 */
void Envelope::calculateMultiplier(double startLevel, double endLevel, unsigned
lengthInSamples) {
  mMultiplier = 1.0 + (std::log(endLevel) - std::log(startLevel)) / lengthInSamples;
}

/**
 * @brief starts the envelope in the attack state
 *
 */
void Envelope::start() {
  // Start the envelope in the attack state
  mState = E_OFF;
  updateState();
}

/**
 * @brief stops the envelope by transitioning into the release state
 *
 */
void Envelope::stop() {
  mState = E_SUSTAIN;
  updateState();
}

/**
```

```
 * @brief Checks if the current state is one that is timed
 *
 * Both off and sustain states can last indefinitely, while the others are timed
 * @return true
 * @return false
 */
bool Envelope::stateIsTimed() {
  return mState != E_OFF && mState != E_SUSTAIN;
}

/**
 * @brief Set all the envelope state values
 *
 * @param attackTime
 * @param decayTime
 * @param sustainLevel
 * @param releaseTime
 */
void Envelope::setEnvelope(double attackTime, double decayTime, double sustainLevel,
double releaseTime) {
  setParameter(ATTACK, attackTime);
  setParameter(DECAY, decayTime);
  setParameter(SUSTAIN, sustainLevel);
  setParameter(RELEASE, releaseTime);
}

/**
 * @brief Set the attck time
 *
 * @param attackTime
 */
void Envelope::setAttackTime(double attackTime) {
  setParameter(ATTACK, attackTime);
}

/**
 * @brief Set the decay time
 *
 * @param decayTime
 */
void Envelope::setDecayTime(double decayTime) {
  setParameter(DECAY, decayTime);
}
```

```
/**
 * @brief Set the release time
 *
 * @param releaseTime
 */
void Envelope::setReleaseTime(double releaseTime) {
  setParameter(DECAY, releaseTime);
}

/**
 * @brief Set the sustain level
 *
 * @param level
 */
void Envelope::setSustainLevel(double level) {
  setParameter(SUSTAIN, level);
}

}
#include "Synth.hpp"
#include <functional>

namespace tsal {


Synth::Synth() {
  mEnvelope.setActive(false);
  mFilter.setMode(Filter::HIGHPASS);
  mLFO.setMode(Oscillator::SINE);
  mLFO.setFrequency(2);
  mLFO.setActive();
}

void Synth::getOutput(AudioBuffer<float> &buffer) {
  if (!mActive) {
    return;
  }
  const auto channels = buffer.getChannelCount();
  const auto frames = buffer.getFrameCount();

  // setChannelPanning(channels);
```

```
  for (unsigned long i = 0; i < frames; i++) {
    const auto envelope = mEnvelope.getSample();
    const auto output = mOscillator.getSample();

    const auto lfoFilterMod = (mLFO.getSample() + 1) / 2;
    mFilter.setCutoff(lfoFilterMod);

    for (unsigned j = 0; j < channels; j++) {
      buffer[i * channels + j] = mFilter.process(output * envelope * (mVelocity / 127.0));
    }
  }
}

/**
 * @brief Start playing a note
 *
 * @param note
 * @param velocity
 */
void Synth::play(double note, double velocity) {
  mNote = note;
  mOscillator.setActive();
  mEnvelope.start();
  setVelocity(velocity);
  mOscillator.setNote(note);
}


/**
 * @brief Stop playing a note. However, this synth is monophonic so the note is irrelavant
 *
 * @param note
 */
void Synth::stop(double note) {
  (void)note;
  if (mEnvelope.isActive()) {
    mEnvelope.stop();
  } else {
    mOscillator.setActive(false);
  }
}

double Synth::getNote() const {
```

```
  return mNote;
}

void Synth::setVelocity(double velocity) {
  mVelocity = Util::checkParameterRange("Synth: Velocity", velocity, mVelocityRange);
}

Util::ParameterRange<double> Synth::mVelocityRange = std::make_pair(0.0, 127.0);

}
#include "Filter.hpp"

namespace tsal {

std::vector<ParameterManager::Parameter> Filter::FilterParameters
({
  { .name="Filter Mode", .min=0.0, .max=2.0, .defaultValue=LOWPASS },
  { .name="Cutoff", .min=0.0, .max=1.0, .defaultValue=0.0, .exclusiveMax=true },
    { .name="Resonance", .min=0.0, .max=1.0, .defaultValue=0.0 }
});

double Filter::process(double input) {
  // https://www.musicdsp.org/en/latest/Filters/131-cascaded-resonant-lp-hp-filter.html
  const double cutoff = getParameter(CUTOFF);
  const double resonance = getParameter(RESONANCE);
  const double cutoffLow = (FilterMode) getParameter(FILTER_MODE) == LOWPASS ?
getParameterData(CUTOFF).max : cutoff;
  const double cutoffHigh = (FilterMode) getParameter(FILTER_MODE) == HIGHPASS ?
getParameterData(CUTOFF).max : cutoff;
  const double fbLow = resonance + resonance/(1 - cutoffLow);
  const double fbHigh = resonance + resonance/(1 - cutoffHigh);
  m1 = m1 + cutoffLow*(input - m1 + fbLow*(m1 - m2)) + mPentium4;
  m2 = m2 + cutoffLow*(m1 - m2);
  m3 = m3 + cutoffHigh*(m2 - m3 + fbHigh*(m3 - m4)) + mPentium4;
  m4 = m4 + cutoffHigh*(m3 - m4);
  return m2 - m4;
}

void Filter::setCutoff(double cutoff) {
  setParameter(CUTOFF, cutoff);
}

void Filter::setResonance(double resonance) {
```

```
  setParameter(RESONANCE, resonance);
}

double Filter::mPentium4 = 1.0e-24;
}
#include "MidiParser.hpp"
#include <iostream>

namespace tsal {

/**
 * @brief Construct a new MidiParser
 *
 */
MidiParser::MidiParser() {
  setNumThreads(1);
}

/**
 * @brief Construct a new MidiParser
 *
 * @param threads
 */
MidiParser::MidiParser(unsigned threads) {
  setNumThreads(threads);
}

/**
 * @brief Construct a new MidiParser and reads in a midi file
 *
 * @param threads
 * @param filename
 */
MidiParser::MidiParser(unsigned threads, const std::string& filename) {
  setNumThreads(threads);
  read(filename);
}

/**
 * @brief Read a midi file from disk
 *
 * A midi file is read and parsed by the midifile library. Useless events are also added
 * to the events stored in memory so that portions of the song can be equally processed
```

```
 * by the number of threads previously specified
 *
 * @param filename
 */
void MidiParser::read(const std::string& filename) {
  if (!validPath(filename)) {
    std::cout << "MidiParser: Invalid path to file: " << filename << std::endl;
    return;
  }
  mMidiFile.read(filename);

  mHasRead = true;
  mMidiFile.joinTracks();
  mMidiFile.linkNotePairs();

  smf::MidiEventList& midiTrack = mMidiFile[0];

  /* Basically, divide up the track time between the number fo threads
   * Each time segment has to identify the midi note that corresponds
   * to its tick.
   */
  std::vector<unsigned> eventRegions;
  int previousRegionBound = 0;
  int totalTicks = mMidiFile.getFileDurationInTicks();
  for (unsigned i = 0; i < mNumThreads; i++) {
    // Set the new bound for the region
    // Equally divided time between threads
    int tick = (i + 1) * totalTicks/mNumThreads;
    for (int j = 0; j < midiTrack.size(); j++) {
      // Find the events that corresponds to the bound time
      if (midiTrack[j].tick >= tick) {
        // If there are multiple notes at the same tick time, make sure to include them all
        if (j < midiTrack.size() - 1 && midiTrack[j + 1].tick == tick) {
          continue;
        }
        // Add the new midi event region
        eventRegions.push_back(j - previousRegionBound);
        previousRegionBound = j;
        break;
      }
    }
  }
```

```
  // Find the biggest region and make sure all other regions match it in midi events size
  unsigned maxRegion = *std::max_element(eventRegions.begin(), eventRegions.end());
  for (unsigned i = 0; i < eventRegions.size(); i++) {
    int tick = i * totalTicks/eventRegions.size();
    for (unsigned j = eventRegions[i]; j < maxRegion; j++) {
      mMidiFile.addNoteOff(0, tick, 0, 21);
    }
  }

  mMidiFile.sortTracks();
}

/**
 * @brief Get a MidiEvent
 *
 * @param aEvent
 * @return const smf::MidiEvent&
 */
const smf::MidiEvent& MidiParser::operator[] (int aEvent) const {
  return mMidiFile[0][aEvent];
}

/**
 * @brief Set a MidiEvent
 *
 * @param aEvent
 * @return smf::MidiEvent&
 */
smf::MidiEvent& MidiParser::operator[] (int aEvent) {
  return mMidiFile[0][aEvent];
}

bool MidiParser::validPath(const std::string& filename) {
  std::ifstream test(filename);
  bool isValid = (bool) test;
  test.close();
  return isValid;
}

}
#include "Effect.hpp"
#include "Channel.hpp"
namespace tsal {
```

```
std::vector<ParameterManager::Parameter> Effect::EffectParameters
({
  { .name="Wet/Dry", .min=0.0, .max=1.0, .defaultValue=1.0 },
 });

Effect::~Effect() {
  if (mParentChannel != nullptr)
    mParentChannel->remove(*this);
}

void Effect::setParentChannel(Channel* channel) {
  if (mParentChannel != nullptr)
    mParentChannel->remove(*this);
  mParentChannel = channel;
}

}
#include "EffectChain.hpp"

namespace tsal {

EffectChain::~EffectChain() {
  for (unsigned i = 0; i < mEffects.size(); i++) {
    remove(*mEffects[i]);
  }
}

void EffectChain::getOutput(AudioBuffer<float> &buffer) {
  for (auto effect : mEffects) {
    effect->getOutput(buffer);
  }
}

void EffectChain::updateContext(const Context& context) {
  OutputDevice::updateContext(context);
  for (auto effect : mEffects) {
    effect->updateContext(context);
  }
}

/* @brief Add an effect to the end of the chain
 *
```

```
 * @param effect
 */
void EffectChain::add(Effect& effect) {
  mContext.requestModelChange(std::bind(&EffectChain::addDeviceToModel, this,
std::ref(effect)));
}

void EffectChain::addDeviceToModel(Effect& effect) {
  effect.updateContext(mContext);
  mEffects.push_back(&effect);
}

/* @brief Remove an effect from the effect chain
 *
 * @param effect
 */
void EffectChain::remove(Effect& effect) {
  mContext.requestModelChange(std::bind(&EffectChain::removeDeviceFromModel, this,
std::ref(effect)));
}

void EffectChain::removeDeviceFromModel(Effect& effect) {
  for (unsigned i = 0; i < mEffects.size(); i++) {
    if (mEffects[i] == &effect) {
      effect.updateContext(Context::clear());
      mEffects.erase(mEffects.begin() + i);
    }
  }
  std::cout << std::endl;
}

}
#include "Delay.hpp"

namespace tsal {

std::vector<ParameterManager::Parameter> Delay::DelayParameters
({
  { .name="Delay", .min=0.0, .max=5.0, .defaultValue=1.0 },
  { .name="Feedback", .min=0.0, .max=1.0, .defaultValue=0.5 }
});

void Delay::updateContext(const Context& context) {
```

```
  OutputDevice::updateContext(context);
  // Update context dependent parameters
  parameterUpdate(DELAY);
  mBuffer.setFrameCount(mDelayFrames);
  mBuffer.setChannelCount(mContext.getChannelCount());
}

void Delay::getOutput(AudioBuffer<float> &buffer) {
  if (!mActive) {
    return;
  }

  const auto channels = buffer.getChannelCount();
  for (unsigned long i = 0; i < buffer.getFrameCount(); i++) {
    int offset = mCounter - mDelayFrames;
    if (offset < 0) offset = mBuffer.size() + offset;

    for (unsigned j = 0; j < channels; j++) {
      const double output = mBuffer[offset];
      const double bufferValue = buffer[i * channels + j] + output *
getParameter(FEEDBACK);
      mBuffer[mCounter++] = bufferValue;
      if (mCounter >= mBuffer.size()) {
        mCounter = 0;
      }
      buffer[i * channels + j] += output;
    }
  }
}

/**
 * @brief Set the delay
 *
 * @param delay (seconds)
 */
void Delay::setDelay(double delay) {
  setParameter(DELAY, delay);
}

/**
 * @brief Set the feedback
 *
 * @param feedback
```

```
 */
void Delay::setFeedback(double feedback) {
  setParameter(FEEDBACK, feedback);
}

void Delay::parameterUpdate(unsigned id) {
  switch(id) {
  case DELAY:
    mDelayFrames = std::round(getParameter(DELAY) * mContext.getSampleRate());
    break;
  }
}

}
#include "Mixer.hpp"
#include <mutex>
#include <portaudio.h>
#include <vector>

namespace tsal {

void Mixer::paStreamFinished(void* userData) {
  (void) userData;
  return;
}

int Mixer::paCallback( const void *inputBuffer, void *outputBuffer,
                       unsigned long framesPerBuffer,
                       const PaStreamCallbackTimeInfo* timeInfo,
                       PaStreamCallbackFlags statusFlags,
                       void *userData ) {
  (void) timeInfo; /* Prevent unused variable warnings. */
  (void) statusFlags;
  (void) inputBuffer;

  Mixer *mixer = static_cast<Mixer *>(userData);
  return mixer->audioCallback((float*) outputBuffer, framesPerBuffer);
}

int Mixer::audioCallback(float *outputBuffer, unsigned long frameCount) {
  mProcessing = true;
  const auto channels = mContext.getChannelCount();
  mBuffer.setSize(frameCount, channels);
```

```
  mBuffer.clear();
  mMaster.getOutput(mBuffer);
  for (unsigned long i = 0; i < frameCount; i++) {
    mSequencer.tick();
    for (unsigned j = 0; j < channels; j++) {
      outputBuffer[i * channels + j] = mBuffer[i * channels + j];
    }
  }
  // Update the model
  runModelChanges();

  return paContinue;
}

void Mixer::openPaStream() {
  PaError err = Pa_Initialize();
  if (err != paNoError) {
    return;
  }
  PaStreamParameters outputParameters;

  const PaDeviceIndex index = Pa_GetDefaultOutputDevice();
  outputParameters.device = index;
  if (outputParameters.device == paNoDevice) {
    printf("No default output device found\n");
    return;
  }

  const PaDeviceInfo* pInfo = Pa_GetDeviceInfo(index);
  if (pInfo != 0) {
    printf("Output device name: '%s'\n", pInfo->name);
    printf("Default sample rate: '%f'\n", pInfo->defaultSampleRate);
  }

  outputParameters.channelCount = mContext.getChannelCount();
  outputParameters.sampleFormat = paFloat32; /* 32 bit floating point output */
  outputParameters.suggestedLatency = Pa_GetDeviceInfo( outputParameters.device
)->defaultLowOutputLatency;
  outputParameters.hostApiSpecificStreamInfo = NULL;

  mSequencer.setSampleRate(mContext.getSampleRate());
  mSequencer.setBPM(60);
  mSequencer.setPPQ(240);
```

```
  // Add a compressor so people don't break their sound cards
  // mMaster.add(mCompressor);
  err = Pa_OpenStream(&mPaStream,
                      NULL, /* no input */
                      &outputParameters,
                      mContext.getSampleRate(),
                      paFramesPerBufferUnspecified,
                      paClipOff,      /* we won't output out of range samples so don't bother
clipping them */
                      &Mixer::paCallback,
                      this
                      );

  if (err != paNoError) {
    /* Failed to open stream to device !!! */
    return;
  }

  err = Pa_SetStreamFinishedCallback(mPaStream, &Mixer::paStreamFinished);

  if (err != paNoError) {
    Pa_CloseStream(mPaStream);
    mPaStream = 0;
    return;
  }
  err = Pa_StartStream(mPaStream);
  if (err != paNoError) {
    return;
  }

  mMaster.updateContext(mContext);
}

Mixer::Mixer() : mContext(44100, 2, this) {
  openPaStream();
}

/**
 * @brief Construct a new Mixer
 *
 * @param sampleRate if default constructor is used, Mixer will default to the highest
sample rate supported
```

```
 */
Mixer::Mixer(unsigned sampleRate) : mContext(sampleRate, 2, this) {
  openPaStream();
}

Mixer::~Mixer() {
  PaError err = Pa_CloseStream(mPaStream);
  if (err != paNoError) {
    return;
  }
  err = Pa_Terminate();
  if (err != paNoError) {
    return;
  }
}

/**
 * @brief Make a change to the DAW model
 * Since the callback function can be accessing any given device in the DAW (channel,
instrument, effect),
 * changes to this model have to be mutually exclusive from the sound device callback
 *
 * @param change a function that is executed safely for making model changes
 */
void Mixer::requestModelChange(std::function<void()> change) {
  // This is currently a pretty naive implementation
  // It assumes that the audio callback function is constantly being run

  // Increment the model change request count
  std::unique_lock<std::mutex> changeLock(mChangeRequestMutex);
  mChangeRequests++;
  changeLock.unlock();

  // Start the critical section for the change
  std::unique_lock<std::mutex> modelLock(mModelMutex);
  mModelChangeRequestCondition.wait(modelLock);

  // Wait until the main thread is waiting
  // Or else we may finish the change before the main thread waits
  std::unique_lock<std::mutex> waitLock(mWaitModelChangeMutex);

  // We can safely modify the model
  change();
```

```
  changeLock.lock();
  bool moreChanges = --mChangeRequests;
  changeLock.unlock();

  // Check if there are no more changes
  if (!moreChanges) {
    mWaitModelChangeCondition.notify_one();
  } else {
    modelLock.unlock();
    mModelChangeRequestCondition.notify_one();
  }
}

/**
 * @brief Run the scheduled changes to the model
 *
 */
void Mixer::runModelChanges() {
  // Check if there are requests to change the model
  if (mChangeRequests) {
    std::unique_lock<std::mutex> lk(mWaitModelChangeMutex);
    mModelChangeRequestCondition.notify_one();
    mWaitModelChangeCondition.wait(lk);
  }
}

/**
 * @brief Add a channel
 *
 * @param channel
 */
void Mixer::add(Channel& channel) {
  mMaster.add(channel);
}

/**
 * @brief Remove a channel
 * @param channel
 */
void Mixer::remove(Channel& channel) {
  mMaster.remove(channel);
}
```

```
/**
 * @brief Add an instrument
 *
 * Add an instrument to the default master channel
 *
 * @param instrument
 */
void Mixer::add(Instrument& instrument) {
  mMaster.add(instrument);
}

/**
 * @brief Remove an instrument
 *
 * Remove an instrument that was added to the default master channel
 *
 * @param instrument
 */
void Mixer::remove(Instrument& instrument) {
  mMaster.remove(instrument);
}

/**
 * @brief Add an effect
 *
 * Add an effect to the default master channel
 *
 * @param effect
 */
void Mixer::add(Effect& effect) {
  mMaster.add(effect);
}


/**
 * @brief Remove an effect
 *
 * Remove an effect that was added to the default master channel
 *
 * @param effect
 */
void Mixer::remove(Effect& effect) {
```

```
  mMaster.remove(effect);
}


}
#include "Oscillator.hpp"
#include "Util.hpp"
#include "MidiNotes.hpp"
#include <iostream>

namespace tsal {

std::vector<ParameterManager::Parameter> Oscillator::OscillatorParameters
({
  { .name="Oscillator Mode", .min=0.0, .max=3.0, .defaultValue=0.0 },
  { .name="Modulation Mode", .min=0.0, .max=3.0, .defaultValue=0.0 },
  { .name="Modulation", .min=-1.0, .max=1.0, .defaultValue=0.0 },
  { .name="Frequency", .min=0.0, .max=15000.0, .defaultValue=1.0 },
  { .name="Phase Offset", .min=0.0, .max=1.0, .defaultValue=0.0 },
});
// Helpful implementation of ployBLEP to reduce aliasing
// http://metafunction.co.uk/all-about-digital-oscillators-part-2-blits-bleps/
double Oscillator::getSample() {
  const double t = mPhase / PI2;
  const double modulation = getParameter(MODULATION);
  const ModulationMode modulationMode = (ModulationMode)
getParameter(MODULATION_MODE);

  double phase = mPhase + (getParameter(PHASE_OFFSET) * PI2);
  double output = 0.0;

  if (modulationMode == PM) {
    phase += modulation;
  }

  switch ((OscillatorMode) getParameter(OSCILLATOR_MODE)) {
    case SINE:
      output = sin(phase);
      break;
    case SAW:
      output = (2.0 * phase / PI2) - 1.0;
      output -= polyBLEP(t); // Layer output of Poly BLEP on top
      break;
```

```
    case SQUARE:
      output = phase < M_PI ? 1 : -1;
      output += polyBLEP(t); // Layer output of Poly BLEP on top (flip)
      output -= polyBLEP(fmod(t + 0.5, 1.0)); // Layer output of Poly BLEP on top (flop)
      break;
    case WHITE_NOISE:
      output = Util::random() * 2 - 1;
      break;
  }

  if (modulationMode == AM) {
    output *= modulation;
  } else if (modulationMode == MIX) {
    output = (output + modulation) / 2.0;
  }


  mPhase += mPhaseStep;
  while (mPhase >= PI2)
    mPhase -= PI2;

  return output;
}

/**
 * @brief Get the note from a corresponding frequency
 *
 * @param frequency
 * @return unsigned
 */
unsigned Oscillator::getNoteFromFrequency(double frequency) {
  return (12/log(2)) * std::log(frequency/27.5) + 21;
}

/**
 * @brief Get the frequency from a corresponding note
 *
 * @param note
 * @return double
 */
double Oscillator::getFrequencyFromNote(double note) {
  return 27.5 * pow(2.0, (note - 21.0) / 12.0);
}
```

```
double Oscillator::polyBLEP(double t)
{
  double dt = mPhaseStep / PI2;

  // t-t^2/2 +1/2
  // 0 < t <= 1
  // discontinuities between 0 & 1
  if (t < dt) {
   t /= dt;
   return t + t - t * t - 1.0;
  }

  // t^2/2 +t +1/2
  // -1 <= t <= 0
  // discontinuities between -1 & 0
  else if (t > 1.0 - dt) {
   t = (t - 1.0) / dt;
   return t * t + t + t + 1.0;
  }

  // no discontinuities
  // 0 otherwise
  else return 0.0;
}

/**
 * @brief Set the note
 *
 * @param note (midi format)
 */
void Oscillator::setNote(double note) {
  setParameter(FREQUENCY, getFrequencyFromNote(note));
}

/**
 * @brief Set the frequncy
 *
 * @param frequency
 */
void Oscillator::setFrequency(double frequency) {
  setParameter(FREQUENCY, frequency);
}
```

```
/**
 * @brief Set the mode
 *
 * @param mode
 */
void Oscillator::setMode(OscillatorMode mode) {
  setParameter(OSCILLATOR_MODE, mode);
}

/**
 * @brief Get the frequency
 *
 * @return double
 */
double Oscillator::getFrequency() {
  return getParameter(FREQUENCY);
}

/**
 * @brief Get the note
 *
 * @return unsigned (midi)
 */
unsigned Oscillator::getNote() {
  return getNoteFromFrequency(getParameter(FREQUENCY));
}

void Oscillator::parameterUpdate(unsigned id) {
  switch (id) {
  case FREQUENCY:
    mPhaseStep = getParameter(FREQUENCY) * PI2 / mContext.getSampleRate();
    break;
  }
}

}
#ifndef CHANNEL_H
#define CHANNEL_H

#include "EffectChain.hpp"
#include "Instrument.hpp"
#include "OutputDevice.hpp"
```

```
#include "ParameterManager.hpp"
#include "Amp.hpp"
#include "Panning.hpp"

namespace tsal {

/** @class Channel
 * @brief Routes instruments and other channels through an effect chain and to the
master channel
 * on the mixer
 *
 * Instruments, effects, and channels can all be added to a channel. The mixer has one
default master channel
 * but more channels can be added to the mixer to increase flexibility
 */
class Channel : public ChannelDevice, public ParameterManager {
  public:
    Channel();
    virtual ~Channel();
    static std::vector<Parameter> ChannelParameters;
    enum Parameters {
                VOLUME = 0,
                PANNING,
    };
    virtual void getOutput(AudioBuffer<float> &buffer) override;
    virtual void updateContext(const Context& context) override;
    virtual void setParentChannel(Channel* channel) override;
    void add(Channel& channel);
    void add(Effect& effect, unsigned position);
    void add(Effect& effect);
    void add(Instrument& instrument);

    void remove(Effect& effect);
    void remove(Channel& channel);
    void remove(Instrument& instrument);

    int getChannelCount() { return mRoutedChannels.size(); };
    int getInstrumentCount() { return mRoutedInstruments.size(); };
    int getEffectCount() { return mEffectChain.size(); };

  protected:
    virtual void parameterUpdate(unsigned int id) override;
  private:
```

```
    Amp mAmp;
    Panning mPanning;
    RouteDevice<OutputDevice> mChannelIn;
    RouteDevice<Instrument> mRoutedInstruments;
    RouteDevice<Channel> mRoutedChannels;
    EffectChain mEffectChain;
};

}

#endif
#ifndef CHANNELDEVICE_H
#define CHANNELDEVICE_H

#include "OutputDevice.hpp"

namespace tsal {

//Forward declaration of Channel
class Channel;

/** @class ChannelDevice
 * @brief Base class for a device that can be routed to a channel
 */
class ChannelDevice : public OutputDevice {
  public:
    ChannelDevice() = default;
  protected:
    virtual void setParentChannel(Channel* channel) = 0;
    Channel* mParentChannel = nullptr;
};

}

#endif
#ifndef OUTPUTDEVICE_H
#define OUTPUTDEVICE_H

#include "Util.hpp"
#include "AudioBuffer.hpp"
#include "Context.hpp"
#include <iostream>
#include <vector>
```

```
namespace tsal {

// Forward declarations
class Mixer;

/** @class OutputDevice
 * @brief Base class for a device that produces audio output
 *
 */
class OutputDevice {
  public:
    OutputDevice() = default;
    virtual ~OutputDevice() {};
    /**
     * @brief Get the output for the device
     *
     * @return double
     */
    virtual void getOutput(AudioBuffer<float>& buffer) { (void)buffer; return; };
    virtual void setActive(bool active = true);
    virtual void updateContext(const Context& context) { mContext.update(context); };

    bool isActive() const;

  protected:
    bool mActive = true;
    Context mContext;
};

}

#endif
#ifndef EFFECTCHAIN_H
#define EFFECTCHAIN_H

#include "OutputDevice.hpp"
#include "Effect.hpp"
#include <functional>

namespace tsal {

/** @class EffectChain
```

```
 * @brief Manages the effect chain for a channel
 *
 * Effects are applied to a channel linked in a chain and updating the audio
 * stream as it passes through the chain
 */
class EffectChain : public OutputDevice {
  public:
    ~EffectChain();
    virtual void getOutput(AudioBuffer<float> &buffer) override;
    virtual void updateContext(const Context& context) override;
    void add(Effect& effect);
    void remove(Effect& effect);
    int size() { return mEffects.size(); };
  private:
    std::vector<Effect*> mEffects;
    void addDeviceToModel(Effect& effect);
    void removeDeviceFromModel(Effect& effect);
};

}

#endif
#ifndef SOUNDFONT_H
#define SOUNDFONT_H

#include "Instrument.hpp"

#ifndef TSF_IMPLEMENTATION
#define TSF_IMPLEMENTATION
#include "tsf.h"


namespace tsal {

/** @class SoundFont
 * @brief A SoundFont 2 synthesizer that can sf2 files
 *
 * SoundFonts seem to be the easiest way to synthesize real instruments. The
 * SoundFont class uses the TinySoundFont library to load and play sf2 files.
 *
 * Useful links:
 * - https://github.com/schellingb/TinySoundFont/
 * - http://www.synthfont.com/soundfonts.html
```

```
 */
class SoundFont : public Instrument {
  public:
    SoundFont(std::string filename) {
      mSoundFont = tsf_load_filename(filename.c_str());
      if (mSoundFont == nullptr) {
        std::cout << "Failed to load SoundFont: " << filename << std::endl;
        return;
      }
    };
    virtual ~SoundFont() {
      tsf_close(mSoundFont);
    };
    virtual void getOutput(AudioBuffer<float> &buffer) override {
      return;
      tsf_render_float(mSoundFont, buffer.getRawPointer(), buffer.size(), 0);
    };
    virtual void play(double note, double velocity = 100.0) override {
      tsf_note_on(mSoundFont, mPresetIndex, note, velocity/127.0);
    };
    virtual void stop(double note) override {
      tsf_note_off(mSoundFont, mPresetIndex, note);
    };
    void channelNoteOn(int channel, unsigned note, double velocity = 100.0) {
    };
    void channelNoteOff(int channel, unsigned note) {};

    void reset() {};
    void setPreset(int presetIndex) {
      mPresetIndex = presetIndex;
    };
    void setPreset(std::string presetName) {
      setPreset(getPresetIndex(presetName));
    };

    int getPresetIndex(int bank, int presetNumber) {
      return tsf_get_presetindex(mSoundFont, bank, presetNumber);
    };
    int getPresetIndex(std::string presetName) {
      for (int i = 0; i < getPresetCount(); i++) {
        if (presetName == getPresetName(i)) {
          return i;
        }
```

```
    }
    return -1;
  };
  int getPresetCount() {
    return tsf_get_presetcount(mSoundFont);
  };
  std::string getPresetName(int presetIndex) {
    return std::string(tsf_get_presetname(mSoundFont, presetIndex));
  };
  std::string getPresetName(int bank, int presetNumber) {
    return std::string(tsf_bank_get_presetname(mSoundFont, bank, presetNumber));
  };
 private:
  tsf* mSoundFont;
  int mPresetIndex = 0;
};

}

#endif

#endif
#ifndef TSAL_H
#define TSAL_H

#include "Channel.hpp"
#include "Compressor.hpp"
#include "Delay.hpp"
#include "Mixer.hpp"
#include "Oscillator.hpp"
#include "PolySynth.hpp"
#include "ProgressOctave.hpp"
// #include "SoundFont.hpp"
#include "Synth.hpp"
#include "ThreadSynth.hpp"
// #include "LadspaManager.hpp"
// #include "LadspaEffect.hpp"
#include "Timing.hpp"

#endif
#ifndef ENVELOPE_H
#define ENVELOPE_H
```

```
#include <algorithm>
#include "Mixer.hpp"
#include "OutputDevice.hpp"
#include "ParameterManager.hpp"
#include "Util.hpp"
#include <cmath>

namespace tsal {

/** @class Envelope
 * @brief An sound envelope that can be used to change a sound over time
 *
 * A 4 stage envelope transitions from attack, to decay, to sustain, and then to release. It
achieves
 * a smoother sound when overlayed on an instrument
 * Implementation adapted from
http://www.martin-finke.de/blog/articles/audio-plugins-011-envelopes/
 */
class Envelope : public OutputDevice, public ParameterManager {
  public:
    Envelope() :
      ParameterManager(EnvelopeParameters) {};
    Envelope(std::vector<Parameter> parameters) :
      ParameterManager(EnvelopeParameters) {
      addParameters(parameters);
    }
    static std::vector<Parameter> EnvelopeParameters;
    enum Parameters {
                ATTACK,
                DECAY,
                SUSTAIN,
                RELEASE,
    };
    double getSample();
    void updateState();
    void start();
    void stop();
    void setAttackTime(double attackTime);
    void setDecayTime(double decayTime);
    void setSustainLevel(double level);
    void setReleaseTime(double releaseTime);
    void setEnvelope(double attackTime, double decayTime, double sustainLevel, double
releaseTime);
```

```
  private:
    enum EnvelopeState {
                    E_ATTACK,
                    E_DECAY,
                    E_SUSTAIN,
                    E_RELEASE,
                    E_OFF,
    };
    void calculateMultiplier(double startLevel, double endLevel, unsigned
lengthInSamples);
    double getStateValue(EnvelopeState state);
    bool stateIsTimed();
    EnvelopeState mState = E_OFF;
    unsigned mCurrentStateIndex = 0;
    unsigned mNextStateIndex = 0;
    double mEnvelopeValue = 0.0;
    double mMultiplier = 0.0;
};

}

#endif
#ifndef AUDIOBUFFER_H
#define AUDIOBUFFER_H

#include <vector>
#include <iostream>

namespace tsal {

/** @class AudioBuffer
 * @brief Stores audio data in an interleaved buffer multiplexed by channels and
frames.
 *
 * With functionality for storing and manipulating audio data, AudioBuffer is
used or interfaces with almost every class in TSAL.
 * It is responsible for routing audio data throughout the system.
 * AudioBuffer stores a series of frames which are made up of some number of
channels.
 * In the case of stereo audio, there may be some number of frames with 2 channels
in each frame.
 * For example, the first 2 values in the stereo buffer will be samples for the
left and right channel which comprise the first frame
```

```
 * The next 2 values are the left and right samples for the second frame and so
on.
 * This method is optimized for CPU spatial locality since buffers are commonily iterated
through sequentially
 */
template <typename T>
class AudioBuffer {
  public:
    AudioBuffer()
      : mFrameCount(0),
        mChannelCount(0) {};
    AudioBuffer(unsigned long frameCount, unsigned channelCount)
      : mBuffer(frameCount * channelCount, 0),
        mFrameCount(frameCount),
        mChannelCount(channelCount) {};
    void setSize(unsigned long frameCount, unsigned channelCount);
    void setSize(AudioBuffer<T>& buffer) { setSize(buffer.getFrameCount(),
buffer.getChannelCount()); };
    void setFrameCount(unsigned long frameCount);
    void setChannelCount(unsigned channelCount);
    /**
     * @brief Zeros out the buffer
     */
    void clear() { std::fill(mBuffer.begin(), mBuffer.end(), 0); };
    unsigned long getFrameCount() { return mFrameCount; };
    unsigned getChannelCount() { return mChannelCount; };
    unsigned long size() { return mFrameCount * mChannelCount; };
    /**
     * @brief Get the raw pointer to the buffer data. Use carefully
     */
    T* getRawPointer() { return mBuffer.data(); };
    const T& operator[](int index) const;
    T& operator[](int index);

    void deinterleaveBuffer(std::vector<AudioBuffer<T>*>& buffers);
    void interleaveBuffers(std::vector<AudioBuffer<T>*>& buffers);
  private:
    void resize();
    std::vector<T> mBuffer;
    unsigned long mFrameCount;
    unsigned mChannelCount;
};
```

```
template <typename T>
const T& AudioBuffer<T>::operator[](int index) const {
  return mBuffer[index];
}

template <typename T>
T& AudioBuffer<T>::operator[](int index) {
  return mBuffer[index];
}

template <typename T>
void AudioBuffer<T>::resize() {
  unsigned long bufferSize = mFrameCount * mChannelCount;
  mBuffer.resize(bufferSize);
}

template <typename T>
void AudioBuffer<T>::setSize(unsigned long frameCount, unsigned channelCount) {
  setFrameCount(frameCount);
  setChannelCount(channelCount);
}

template <typename T>
void AudioBuffer<T>::setFrameCount(unsigned long frameCount) {
  mFrameCount = frameCount;
  // Resize the buffer with the new frame count
  resize();
}

template <typename T>
void AudioBuffer<T>::setChannelCount(unsigned channelCount) {
  mChannelCount = channelCount;
  // Resize the buffer with the new channel count
  resize();
}

/**
 * @brief Combine multiple buffers into one buffer
 *
 * @param buffers the group of buffers of interleave
 *
 * When there are multiple buffers, like one for each channel, this function interleaves
multiple buffers into one buffer.
```

```
 */
template <typename T>
void AudioBuffer<T>::interleaveBuffers(std::vector<AudioBuffer<T>*>& buffers) {
  if (buffers.size() == 0) {
    // Maybe these should throw and exception
    std::cout << "Cannot interleave buffers of size 0" << std::endl;
    return;
  }
  if (mChannelCount % buffers.size() != 0) {
    std::cout << "Cannot demultiplex buffers into channels" << std::endl;
    return;
  }

  const unsigned channelDemultiplex = mChannelCount / buffers.size();

  setFrameCount(buffers[0]->getFrameCount());

  for (unsigned long i = 0; i < mFrameCount; i++) {
    for (unsigned j = 0; j < mChannelCount; j += channelDemultiplex) {
      for (unsigned k = 0; k < channelDemultiplex; k++) {
        mBuffer[i * mChannelCount + j + k] = (*buffers[j])[i];
      }
    }
  }
}

/**
 * @brief Create multiple buffers from one buffer
 *
 * @param buffers the group of buffers to deinterleave into
 *
 * When there are multiple channels interleaved in one buffer, this function deinterleaves
the one buffer into multiple, one for each channel.
 */
template <typename T>
void AudioBuffer<T>::deinterleaveBuffer(std::vector<AudioBuffer<T>*>& buffers) {
  if (buffers.size() > mChannelCount) {
    std::cout << "Cannot interleave buffers of size " << buffers.size() << std::endl;
    return;
  }
  if (mChannelCount % buffers.size() != 0) {
    std::cout << "Cannot demultiplex buffers into channels" << std::endl;
  }
```

```
  const unsigned channelMultiplex = mChannelCount / buffers.size();

  for (auto buffer : buffers) {
    buffer->setChannelCount(1);
    buffer->setFrameCount(mFrameCount);
  }

  for (unsigned long i = 0; i < mFrameCount; i++) {
    for (unsigned j = 0; j < mChannelCount; j += channelMultiplex) {
      AudioBuffer<T>& b= (*buffers[j]);
      b[i] = 0;
      for (unsigned k = 0; k < channelMultiplex; k++) {
        b[i] += mBuffer[i * mChannelCount + j + k];
      }
      b[i] /= channelMultiplex;
    }
  }
}

}

#endif
#ifndef UTIL_H
#define UTIL_H

#include "Timing.hpp"
#include <iostream>
#include <algorithm>
#include <random>
#define _USE_MATH_DEFINES
#include <cmath>
#define PI2 (M_PI * 2)

#include <chrono>
#include <thread>

namespace tsal {

/** @class Util
 * @brief A utilities class
 *
 * Provides basic shared utility functions that programs and classes
```

```
 * can make use of.
 */
class Util {
  public:
    static double random() { return mDist(mGen); };
    static double ampToDb(double amplitude);
    static double dbToAmp(double db);
    static double volumeToDb(double volume);
    static double dbToVolume(double db);
    static void thread_sleep(unsigned duration, Timing::TimeScale scale =
Timing::MILLISECOND);

    template <typename T>
    using ParameterRange = std::pair<T, T>;

    template <typename T>
    static T checkParameterRange(const std::string& name, T value, ParameterRange<T>
range) {
      if (value < range.first || value > range.second) {
        std::cout << name << ": not in range["
                  << range.first << ", "
                  << range.second
                  << "]" << std::endl;
      }
      return std::min(std::max(value, range.first), range.second);
    }

    /**
     * @brief Forces a hidden floor value on the range
     *
     * Lets the user set expected values without causing divide by 0 or other arithmetic
errors
     */
    template <typename T>
    static T checkParameterRangeHiddenFloor(const std::string& name, T value,
ParameterRange<T> range, T floor) {
      return std::max(Util::checkParameterRange(name, value, range), floor);
    }

    /**
     * @brief Forces a hidden ceiling value on the range
     *
```

```
    * Lets the user set expected values without causing divide by 0 or other arithmetic
errors
     */
    template <typename T>
    static T checkParameterRangeHiddenCeiling(const std::string& name, T value,
ParameterRange<T> range, T ceiling) {
      return std::min(Util::checkParameterRange(name, value, range), ceiling);
    }
    static std::mt19937 mGen;
    static std::uniform_real_distribution<> mDist;
};


}

#endif
#ifndef PANNING_H
#define PANNING_H

#include "OutputDevice.hpp"
#include "ParameterManager.hpp"
#include <vector>

namespace tsal {

/** @class
 * @brief
 */
class Panning {
  public:
    static ParameterManager::Parameter PanningParameter;
    double getPanningChannel(unsigned channel) const { return
mChannelPanning[channel]; };
    void setPanning(double panning) { mPanning = panning; };
    void setChannelPanning(unsigned channelCount);
  protected:
    double mPanning = 0.0;
    std::vector<double> mChannelPanning;
  private:
    static double panningConstant;
};

}
```

```
#endif
#ifndef AMP_H
#define AMP_H

#include "Util.hpp"
#include "ParameterManager.hpp"

namespace tsal {

class Amp {
  public:
    static ParameterManager::Parameter AmpParameterVolume;
    static ParameterManager::Parameter AmpParameterGain;
    inline double getAmp() const { return mAmp; };
    void setGain(double gain) {
      mAmp = Util::dbToAmp(gain);
    };
    void setVolume(double volume){
      mAmp = Util::dbToAmp(Util::volumeToDb(volume));
    };
  private:
    double mAmp = 0.5;
};

}
#endif
#ifndef DELAY_H
#define DELAY_H

#include "AudioBuffer.hpp"
#include "ParameterManager.hpp"
#include "Effect.hpp"
#include "Util.hpp"
#include <vector>

namespace tsal {

/** @class Delay
 * @brief A delay effect
 *
 * Adds a delay effect to audio input to achieve a more spacious sound
 */
```

```
class Delay : public Effect {
  public:
    Delay() :
      Effect(DelayParameters) {};
    Delay(std::vector<Parameter> parameters) :
      Effect(DelayParameters) {
      addParameters(parameters);
    };
    static std::vector<Parameter> DelayParameters;
    enum Parameters {
                WET_DRY=0,
                DELAY,
                FEEDBACK,
    };
    virtual void updateContext(const Context& context) override;
    virtual void getOutput(AudioBuffer<float> &buffer) override;
    void setDelay(double delay);
    void setFeedback(double feedback);
  protected:
    virtual void parameterUpdate(unsigned id) override;
  private:
    AudioBuffer<float> mBuffer;
    unsigned mCounter = 0;
    unsigned mDelayFrames = 0;
};

}
#endif
#ifndef THREADSYNTH_H
#define THREADSYNTH_H

#include "Sequencer.hpp"
#include "Synth.hpp"
#include <condition_variable>

namespace tsal {

/** @class ThreadSynth
 * @brief A threaded audio synthesizer
 *
 * A more specific implementation of the Synth class that dedicates the synth's
 * thread as the means to schedule events and timings for notes. This class
 * is mainly used for multithreading examples and doesn't really have
```

```
 * any other meaningful applications
 */
class ThreadSynth : public Synth {
  public:
    virtual void play(double note, double velocity) override {
      Synth::play(note, velocity);
    };
    virtual void stop(double note = MidiNote::A0) override {
      Synth::stop();
    };
    void play(double note, Timing::TimeScale scale, unsigned duration);
    void stop(Timing::TimeScale scale, unsigned multiplier);
};

}

#endif
#ifndef TIMING_H
#define TIMING_H

namespace tsal {

class Timing {
  public:
    enum TimeScale {
      SECOND = 1000000,
      MILLISECOND = 1000,
      MICROSECOND = 1,
    };
    enum NoteScale {
      TICK = 0,
      EIGHTH = 8,
      QUATER = 4,
      HALF = 2,
      WHOLE = 1,
    };
};

} // namespace tsal

#endif
#ifndef INPUTDEVICE_H
#define INPUTDEVICE_H
```

```
#include "OutputDevice.hpp"
#include <vector>

namespace tsal {

/** @class InputDevice
 * @brief Base class for a device that takes input
 *
 */
class InputDevice {
  public:
    virtual ~InputDevice() {};
    virtual double getInput() { return 0; };
    // virtual void getInput(AudioBuffer<float> &buffer) { return; };
};

}

#endif
#ifndef MIDIPARSER_H
#define MIDIPARSER_H

// #include "MidiFile.h"
#include "MidiFile.h"
#include <algorithm>

namespace tsal {

/** @class MidiParser
 * @brief Parse a midi file for the sake of multithreading playing
 *
 * Midi files are composed of midi events such as note on and note off. The midifile
library
 * parses the file and extracts these messages into objects. The goals of TSAL are to
 * audiolize multithreading with sound. One example is processing a midi file. Instead of
using different
 * tracks, TSAL overlays sound with multiple threads by reading different portions of a
midi file.
 * If a midi file is prepared with this process in mind, one track can contain multiple parts
by
 * partitioning equally across the length of the track. To make the use of this concept
more felxible,
```

```
 * MidiParser ensures that the number of midi events are equally distributed so that this process can
 * be easily performed. It simply adds useless midi events where necessary based upon the number of threads
 * specified
 *
 */
class MidiParser {
  public:
    MidiParser();
    MidiParser(unsigned threads);
    MidiParser(unsigned threads, const std::string& filename);
    void setNumThreads(unsigned threads) { mNumThreads = threads; };
    void read(const std::string& filename);
    const smf::MidiEvent& operator[] (int aEvent) const;
       smf::MidiEvent& operator[] (int aEvent);
    unsigned size() { return mMidiFile[0].size(); };
    unsigned getPPQ() const { return mMidiFile.getTicksPerQuarterNote(); };

  private:
    bool validPath(const std::string& filename);
    unsigned mNumThreads = 1;
    bool mHasRead = false;
    smf::MidiFile mMidiFile;
};

}

#endif
#ifndef EFFECT_H
#define EFFECT_H

#include "InputDevice.hpp"
#include "OutputDevice.hpp"
#include "RouteDevice.hpp"
#include "Instrument.hpp"
#include "ParameterManager.hpp"
#include <algorithm>

namespace tsal {

/** @class Effect
```

```
 * @brief An audio processing device that can be chained to a channel to produce audio
effects
 * on one or more instruments routed to the channel
 *
 * An effect recieves input from the ChannelIn device or another effect in the chain.
 * It processes the audio and then continues to pass it up the chain until the audio
reaches
 * the end and it routed to the mixer
 */
class Effect : public ChannelDevice, public ParameterManager {
  public:
    Effect() :
      ParameterManager(EffectParameters) {};
    Effect(std::vector<Parameter> parameters) :
      ParameterManager(EffectParameters) {
      addParameters(parameters);
    };
    ~Effect();
    enum Parameters {
                WET_DRY = 0,
    };
    static std::vector<Parameter> EffectParameters;
    virtual void setParentChannel(Channel* channel) override;
};

}

#endif
#ifndef POLYSYNTH_H
#define POLYSYNTH_H

#include "Synth.hpp"
#include "Oscillator.hpp"
#include "Instrument.hpp"
#include "RouteDevice.hpp"

#define NUM_VOICES 10

namespace tsal {

/** @class PolySynth
 * @brief A polyphonic synthesizer
 *
```

```
 * PolySynth uses multiple Synths to create a polyphonic
 * synthesizer that can handle multiple notes being played
 * at the same time
 */
class PolySynth : public Instrument {
  public:
    PolySynth() :
      Instrument(PolySynthParameters),
      mVoices(NUM_VOICES, Voice()) {
      for (auto& voice : mVoices) {
        voice.setActive(false);
      }
      connectParameters();
      mLFO.setMode(Oscillator::SINE);
      mLFO.setActive();
    };
    PolySynth& operator=(const PolySynth& synth) {
      if (this == &synth) return *this;
      mVoices = synth.mVoices;
      mLFO = synth.mLFO;
      connectParameters();
      return *this;
    };
    PolySynth(const PolySynth& synth) : PolySynth() {
      mVoices = synth.mVoices;
      mLFO = synth.mLFO;
      connectParameters();
    };
    PolySynth(const PolySynth&& synth) : PolySynth() {
      mVoices = synth.mVoices;
      mLFO = synth.mLFO;
      connectParameters();
    }
    static std::vector<Parameter> PolySynthParameters;
    enum Parameters {
                VOLUME = 0,
                PANNING,
                OSC1_MODE,
                OSC2_MODE,
                OSC2_OFFSET,
                MODULATION_MODE,
                ENV_ATTACK,
                ENV_DECAY,
```

```
                ENV_SUSTAIN,
                ENV_RELEASE,
                LFO_ACTIVE,
                LFO_MODE,
                LFO_FREQUENCY,
                LFO_ATTACK,
                LFO_DECAY,
                LFO_SUSTAIN,
                LFO_RELEASE,
  };
  virtual void getOutput(AudioBuffer<float> &buffer) override;
  void play(double note, double velocity = 100.0) override;
  void stop(double note) override;
  void setMode(Oscillator::OscillatorMode mode);
  virtual void updateContext(const Context& context) override;
 protected:
  virtual void parameterUpdate(unsigned id) override {
   Instrument::parameterUpdate(id);
   switch(id) {
   case LFO_FREQUENCY:
    mLFO.setFrequency(getParameter(LFO_FREQUENCY));
    break;
   }
  };
 private:
  void connectParameters() {
   mLFO.connectParameter(Oscillator::OSCILLATOR_MODE,
getParameterPointer(LFO_MODE));
   for (auto& voice : mVoices) {
    voice.mOsc1.connectParameter(Oscillator::OSCILLATOR_MODE,
getParameterPointer(OSC1_MODE));
    voice.mOsc2.connectParameter(Oscillator::OSCILLATOR_MODE,
getParameterPointer(OSC2_MODE));
    voice.mOsc1.connectParameter(Oscillator::MODULATION_MODE,
getParameterPointer(PolySynth::MODULATION_MODE));
    voice.mOsc2.connectParameter(Oscillator::PHASE_OFFSET,
getParameterPointer(OSC2_OFFSET));

    voice.mEnvelope.connectParameter(Envelope::ATTACK,
getParameterPointer(ENV_ATTACK));
    voice.mEnvelope.connectParameter(Envelope::DECAY,
getParameterPointer(ENV_DECAY));
```

```
      voice.mEnvelope.connectParameter(Envelope::SUSTAIN,
getParameterPointer(ENV_SUSTAIN));
      voice.mEnvelope.connectParameter(Envelope::RELEASE,
getParameterPointer(ENV_RELEASE));

      voice.mLFOEnvelope.connectParameter(Envelope::ATTACK,
getParameterPointer(LFO_ATTACK));
      voice.mLFOEnvelope.connectParameter(Envelope::DECAY,
getParameterPointer(LFO_DECAY));
      voice.mLFOEnvelope.connectParameter(Envelope::SUSTAIN,
getParameterPointer(LFO_SUSTAIN));
      voice.mLFOEnvelope.connectParameter(Envelope::RELEASE,
getParameterPointer(LFO_RELEASE));
    }
  }
  class Voice : public Instrument {
   public:
    Voice() {
     mEnvelope.setActive();
     mLFOEnvelope.setActive();
     mFilter.setMode(Filter::BANDPASS);
     mFilter.setCutoff(0.1);
    }
    double getSample(double lfo);
    void play(double note, double velocity) override;
    void stop(double note = MidiNote::A0) override;
    virtual void updateContext(const Context& context) override;
    double getNote() { return mOsc1.getNote(); };

    Oscillator mOsc1;
    Oscillator mOsc2;
    Filter mFilter;
    Envelope mEnvelope;
    Envelope mLFOEnvelope;

    Oscillator::ModulationMode mModulationMode = Oscillator::MIX;
    double mVelocity = 100.0;
    unsigned mNote = 0.0;
  };
  Voice* getInactiveVoice();
  Oscillator mLFO;
  std::vector<Voice> mVoices;
};
```

```
}

#endif
#ifndef INSTRUMENT_H
#define INSTRUMENT_H

#include "ChannelDevice.hpp"
#include "ParameterManager.hpp"
#include "Sequencer.hpp"
#include "Panning.hpp"
#include "Amp.hpp"

namespace tsal {

/** @class Instrument
 * @brief An abstract class to provide expected behaviors of instruments
 *
 */
class Instrument : public ChannelDevice, public ParameterManager {
  public:
    Instrument() :
      ParameterManager(InstrumentParameters) {};
    Instrument(std::vector<Parameter> parameters) :
      ParameterManager(InstrumentParameters) {
      addParameters(parameters);
    };
    static std::vector<Parameter> InstrumentParameters;
    enum Parameters {
                VOLUME = 0,
                PANNING,
    };
    virtual void play(double note, double velocity = 100.0) = 0;
    virtual void stop(double note) = 0;
    virtual ~Instrument();
    virtual void setParentChannel(Channel* channel) override;
  protected:
    virtual void parameterUpdate(unsigned id) override;
    Panning mPanning;
    Amp mAmp;
};

}
```

```
#endif
#ifndef MIDINOTE_H
#define MIDINOTE_H

namespace tsal {
enum MidiNote {
  A0 = 21,
  As0,
  B0,
  C1,
  Cs1,
  D1,
  Ds1,
  E1,
  F1,
  Fs1,
  G1,
  Gs1,
  A1,
  As1,
  B1,
  C2,
  Cs2,
  D2,
  Ds2,
  E2,
  F2,
  Fs2,
  G2,
  Gs2,
  A2,
  As2,
  B2,
  C3,
  Cs3,
  D3,
  Ds3,
  E3,
  F3,
  Fs3,
  G3,
  Gs3,
```

A3,
As3,
B3,
C4,
Cs4,
D4,
Ds4,
E4,
F4,
Fs4,
G4,
Gs4,
A4,
As4,
B4,
C5,
Cs5,
D5,
Ds5,
E5,
F5,
Fs5,
G5,
Gs5,
A5,
As5,
B5,
C6,
Cs6,
D6,
Ds6,
E6,
F6,
Fs6,
G6,
Gs6,
A6,
As6,
B6,
C7,
Cs7,
D7,
Ds7,

```
  E7,
  F7,
  Fs7,
  G7,
  Gs7,
  A7,
  As7,
  B7,
  C8
};
}

#endif
#ifndef CONTEXT_H
#define CONTEXT_H

#include <functional>
#include <iostream>

namespace tsal {

class Mixer;

/** @class Context
 * @brief Manages the connection between the Mixer and other devices
 *
 * Context provides a safe interface for devices to interact with the necessary
Mixer methods.
 * An empty can also be used to initialize a device without being connected to a Mixer.
 */
class Context {
  public:
    Context();
    Context(unsigned sampleRate, unsigned channelCount, Mixer *mixer);
    unsigned getChannelCount() const { return mChannelCount; };
    unsigned getSampleRate() const { return mSampleRate; };
    void update(const Context& context) {
      mSampleRate = context.getSampleRate();
      mChannelCount = context.getChannelCount();
      mMixer = context.mMixer;
    };
    static Context clear() {
      return Context();
```

```
    }
    void requestModelChange(std::function<void()> change);
  private:
    unsigned mSampleRate;
    unsigned mChannelCount;
    Mixer *mMixer;
};

}

#endif
#ifndef FILTER_H
#define FILTER_H

#include "Util.hpp"
#include "ParameterManager.hpp"

namespace tsal {

/** @class Filter
 * @brief Applies a filter (low, high, band) to given audio sample
 */
class Filter : public ParameterManager {
  public:
    Filter() :
      ParameterManager(FilterParameters) {};
    Filter(std::vector<Parameter> parameters) :
      ParameterManager(FilterParameters) {
      addParameters(parameters);
    }

    static std::vector<Parameter> FilterParameters;
    enum Parameters {
                FILTER_MODE=0,
                CUTOFF,
                RESONANCE,
    };
    enum FilterMode {
      LOWPASS=0,
      HIGHPASS,
      BANDPASS
    };
```

```
    double process(double input);

    void setMode(FilterMode mode) { mMode = mode; };
    void setCutoff(double cutoff);
    void setResonance(double resonance);

  private:
    FilterMode mMode = LOWPASS;

    double m1 = 0.0;
    double m2 = 0.0;
    double m3 = 0.0;
    double m4 = 0.0;
    static double mPentium4;

};

}
#endif
#ifndef PROGRESSOCTAVE_H
#define PROGRESSOCTAVE_H

#include "Oscillator.hpp"
#include "Instrument.hpp"
#include "RouteDevice.hpp"
#include <vector>

namespace tsal {

/** @class ProgressOctave
 * @brief An instrument that is the equivalent of a progress bar
 *
 * A very basic instrument that when given a starting note, problem size, and number
 * of workers, outputs audio based upon the calculated progress.
 */
class ProgressOctave : public Instrument {
  public:
    ProgressOctave(unsigned startNote, unsigned problemSize, unsigned numWorkers);
    void getOutput(AudioBuffer<float> &buffer) override;
    virtual void updateContext(const Context& context) override;
    void update(unsigned id);
  private:
    unsigned mStartNote;
```

```
   unsigned mProblemSize;
   unsigned mNumWorkers;

   std::vector<std::unique_ptr<Oscillator>> mOscillators;
   RouteDevice<Oscillator> mRoutedOscillators;
   double mStepValue;
};

}

#endif
#ifndef PARAMETERMANAGER_H
#define PARAMETERMANAGER_H

#include <vector>
#include <iostream>
#include <algorithm>
#include <cmath>


#define EXCLUSIVE 0.0001

namespace tsal {

/** @class ParameterManager
 * @brief An interface for defining and modifing parameters on a device
 *
 * ParameterManager offers a interface for defining the parameters and their limits on an
audio device
 */
class ParameterManager {
  public:
   struct Parameter {
      static Parameter copy(const Parameter& parameter, std::string newName) {
        return {
              .name = newName,
              .min = parameter.min,
              .max = parameter.max,
              .defaultValue = parameter.defaultValue,
              .exclusiveMax = parameter.exclusiveMax,
              .exclusiveMin = parameter.exclusiveMin,
        };
      };
```

```
      // The name the of the parameter
      std::string name;
      // The minimum value that a user can assign
      double min;
      // The maximum value that a user can assign
      double max;
      // The default value that the parameter is initialized with
      double defaultValue;
      // Allows value to approach minimum without actually equaling minimum
      bool exclusiveMax = false;
      // Allows value to approach maximum without actually equaling maximum
      bool exclusiveMin = false;
      // Pointer to value
      double* connection = NULL;
    };
    ParameterManager() = default;
    ParameterManager(std::vector<Parameter> parameters) {
      mParameters.swap(parameters);
      for (auto& param : mParameters) {
        initializeParameter(param);
      }
    };

    std::string getParameterName(unsigned id) const { return mParameters[id].name; };
    Parameter& getParameterData(unsigned id) { return mParameters[id]; };
    double* getParameterPointer(unsigned id) { return &(mValues[id]); };
    template <typename T>
    inline T getParameter(unsigned id) { return static_cast<T>(getParameter(id)); };
    inline double getParameter(unsigned id) {
      return mParameters[id].connection == NULL ? mValues[id] :
getConnectedParameter(id);
    };
    inline void setParameter(unsigned id, double value) {
      mValues[id] = std::min(std::max(value, mParameters[id].min), mParameters[id].max);
      parameterUpdate(id);
    }
    /* @brief Connect the parameter to some value
     *
     * @param id identifier for parameter
     * @param connection the pointer for the parameter to reference
     *
     * Rather than setting the value of a parameter, simply connect a parameter
to a pointer
```

```
    * When the getParameter is called, the parameter will update its value from
the pointer
    * This can be used to link parameters together
    */
   void connectParameter(unsigned id, double* connection) {
     mParameters[id].connection = connection;
   }
   /* @brief Disconnect the parameter from some value
    *
    * @param id identifier for parameter
    * @param connection the pointer to disconnect from
    *
    * Returns the parameter to the default behavior
    */
   void disconnectParameter(unsigned id) {
     mParameters[id].connection = NULL;
   }
   void disconnectParameters() {
     for (unsigned i = 0; i < mParameters.size(); i++) {
       disconnectParameter(i);
     }
   }
   size_t getParameterCount() { return mParameters.size(); };
 protected:
   /* @brief a method used as a callback when a parameter is set
    *
    * param id the identifier of the parameter that was just set
    */
   virtual void parameterUpdate(unsigned id) {};
   /* @brief Used to compose parameters when inheriting from a parent class
    *
    * @param parameters the parameters to add to the ParameterManager
    */
   void addParameters(std::vector<Parameter> parameters) {
     for (unsigned i = 0; i < parameters.size(); i++) {
       mParameters.push_back(parameters[i]);
       initializeParameter(mParameters.back());
     }
   }
 private:
   void initializeParameter(Parameter& parameter) {
     if (parameter.exclusiveMax) {
       parameter.max -= EXCLUSIVE;
```

```
    }
    if (parameter.exclusiveMin) {
      parameter.min += EXCLUSIVE;
    }
    mValues.push_back(parameter.defaultValue);
    setParameter(mValues.size() - 1, parameter.defaultValue);
  }
  double getConnectedParameter(unsigned id) {
    return *(mParameters[id].connection);
  };
  std::vector<Parameter> mParameters;
  std::vector<double> mValues;
};

}
#endif
#ifndef COMPRESSOR_H
#define COMPRESSOR_H

#include "Effect.hpp"
#include "Util.hpp"
#include "ParameterManager.hpp"
#include <vector>

/*
 * larger buffer = better quality but more latency
 * I just chose an arbitrary size
 */
#define COMPRESSOR_MAX_BUFFER 512

namespace tsal {

/** @class Compressor
 * @brief An audio compressor
 *
 * A compressor used to reduce clipping. Essentially, if the sound routed through the
 * compressor is louder than the compressors threshold, it will reduce the loudness.
 * Implementation adapted from
https://christianfloisand.wordpress.com/2014/06/09/dynamics-processing-compressorlim
iter-part-1/
 */
class Compressor : public Effect {
  public:
```

```
    Compressor() :
      Effect(CompressorParameters),
      mEnvelope(COMPRESSOR_MAX_BUFFER) {};
    Compressor(std::vector<Parameter> parameters) :
      Effect(CompressorParameters),
      mEnvelope(COMPRESSOR_MAX_BUFFER) {
      addParameters(parameters);
    };
    static std::vector<Parameter> CompressorParameters;
    enum Parameters {
                WET_DRY=0,
                ATTACK,
                RELEASE,
                THRESHOLD,
                RATIO,
                PRE_GAIN,
                POST_GAIN,
    };
    virtual void getOutput(AudioBuffer<float>& buffer) override;
    virtual void updateContext(const Context& context) override;
    void setAttackTime(double attackTime);
    void setReleaseTime(double releaseTime);
    void setThreshold(double threshold);
    void setRatio(double ratio);
    void setPreGain(double preGain);
    void setPostGain(double postGain);
  protected:
    virtual void parameterUpdate(unsigned id) override;
  private:
    void filterAudio(AudioBuffer<float>& buffer);
    void getEnvelope(AudioBuffer<float>& buffer);
    void calculateSlope();
    double mEnvelopeSample = 0.0;

    std::vector<float> mEnvelope;

    double mSlope = 0.0;
    double mGain = 0.0;
    double mAttackGain = 0.0;
    double mReleaseGain = 0.0;
};

}
```

```
#endif
#ifndef ROUTEDEVICE_H
#define ROUTEDEVICE_H

#include "OutputDevice.hpp"
#include <mutex>
#include <vector>
#include <memory>
#include <functional>

namespace tsal {

/** @class RouteDevice
 * @brief Routes any number of inputs into a single output
 *
 * RouteDevice is a flexible class intended to make managing inputs easier.
 * Combining multiple inputs is as simple as adding up all the input values.

*/
template <typename DeviceType>
class RouteDevice : public OutputDevice {
  public:
    RouteDevice() = default;
    ~RouteDevice();
    virtual void getOutput(AudioBuffer<float> &buffer) override;
    virtual void updateContext(const Context& context) override;

    void add(DeviceType& device);
    void remove(DeviceType& device);
    int size() { return mRoutedInputs.size(); };
    DeviceType& operator[](const int index);
    const DeviceType& operator[](const int index) const;

    // std::vector<DeviceType*>& getInputDevices() { return mRoutedInputs; };

  protected:
    void addDeviceToModel(DeviceType& device);
    void removeDeviceFromModel(DeviceType& device);
    /* Store an input device with an output buffer
     * Each device buffer is combined/mixed within this class
     * so that they aren't modifying existing data
     */
    struct RoutedInput {
```

```
      RoutedInput(DeviceType* d)
        : device(d) {};
      DeviceType* device;
      AudioBuffer<float> buffer;
    };
    bool outOfRange(const int index) const;
    std::vector<std::unique_ptr<RoutedInput>> mRoutedInputs;
};


template <typename DeviceType>
RouteDevice<DeviceType>::~RouteDevice() {
  for (unsigned i = 0; i < mRoutedInputs.size(); i++) {
    mRoutedInputs.erase(mRoutedInputs.begin() + i);
  }
}

template <typename DeviceType>
void RouteDevice<DeviceType>::getOutput(AudioBuffer<float> &buffer) {
  for (const auto& routedInput : mRoutedInputs) {
    routedInput->buffer.setSize(buffer);
    routedInput->buffer.clear();
    routedInput->device->getOutput(routedInput->buffer);
    for (unsigned i = 0; i < buffer.size(); i++) {
      buffer[i] += routedInput->buffer[i];
    }
  }
};

template <typename DeviceType>
void RouteDevice<DeviceType>::updateContext(const Context& context) {
  OutputDevice::updateContext(context);
  for (const auto& routedInput : mRoutedInputs) {
    routedInput->device->updateContext(context);
  }
}

/**
 * @brief Add an output device as input
 *
 * @param output
 */
template <typename DeviceType>
```

```
void RouteDevice<DeviceType>::add(DeviceType& device) {

mContext.requestModelChange(std::bind(&RouteDevice<DeviceType>::addDeviceToMod
el, this, std::ref(device)));
}

template <typename DeviceType>
void RouteDevice<DeviceType>::addDeviceToModel(DeviceType& device) {
  device.updateContext(mContext);
  mRoutedInputs.push_back(std::make_unique<RoutedInput>(&device));
}

/**
 * @brief Remove an output device
 *
 * @param output
 */
template <typename DeviceType>
void RouteDevice<DeviceType>::remove(DeviceType& device) {

mContext.requestModelChange(std::bind(&RouteDevice<DeviceType>::removeDeviceFro
mModel, this, std::ref(device)));
}

template <typename DeviceType>
void RouteDevice<DeviceType>::removeDeviceFromModel(DeviceType& device) {
  for (unsigned i = 0; i < mRoutedInputs.size(); i++) {
    if (&device == mRoutedInputs[i]->device) {
      device.updateContext(Context::clear());
      mRoutedInputs.erase(mRoutedInputs.begin() + i);
      break;
    }
  }
}

template <typename DeviceType>
DeviceType& RouteDevice<DeviceType>::operator[](const int index) {
  if (outOfRange(index)) {
    throw "RouteDevice: Index out of range";
  }
  return *(mRoutedInputs[index]->device);
}
```

```
template <typename DeviceType>
const DeviceType& RouteDevice<DeviceType>::operator[](const int index) const {
  if (outOfRange(index)) {
    throw "RouteDevice: Index out of range";
  }
  return *(mRoutedInputs[index]->device);
}

template <typename DeviceType>
bool RouteDevice<DeviceType>::outOfRange(const int index) const {
  return index < 0 || index > (int) mRoutedInputs.size();
}

} // namespace tsal
#endif
#ifndef MIDISEQUENCER_H
#define MIDISEQUENCER_H

#include "Util.hpp"
#include <condition_variable>
#include <functional>
#include <vector>
#include <algorithm>
#include <queue>

namespace tsal {

// Forward declaration of Mixer
class Mixer;

/** @class Sequencer
 * @brief A sequencer that handles the scheduling of events
 *
 * Sequencer handles the real-time scheduling of the process
 * based upon BPM and PPQ parameters
 */
class Sequencer {
  public:
    void tick();
    void start();
    void stop();
    void setSampleRate(unsigned sampleRate) { mSampleRate = sampleRate; };
    void setTick(unsigned tick);
```

```
    void setBPM(unsigned bpm);
    void setPPQ(unsigned ppq);
    void schedule(std::function<void ()> callback, Timing::NoteScale scale, unsigned
duration = 1);
    unsigned getTick() const;
    unsigned getTicksInNote(Timing::NoteScale note) const;
    /*
     * A helper class that wraps a tick value and callback for event scheduling
     */
    struct Event {
        Event(unsigned t, std::function<void ()> f) {
          tick = t;
          callback = f;
        }
        // A compare function for the sorting of the priority queue
        struct compare {
            bool operator()(const std::unique_ptr<Event>& lhs, const
std::unique_ptr<Event>& rhs) {
              return lhs->tick > rhs->tick;
            }
        };
        unsigned tick;
        std::function<void ()> callback;
    };

  private:
    std::priority_queue<std::unique_ptr<Event>, std::vector<std::unique_ptr<Event>>,
Event::compare> mEventQueue;
    void setSamplesPerTick();
    unsigned mSampleTime = 0;
    unsigned mTick = 0;


    unsigned mPPQ = 240;
    unsigned mBPM = 100;
    unsigned mSampleRate = 0;
    double mSamplesPerTick = 0.0;

    static Util::ParameterRange<unsigned> mBPMRange;
    static Util::ParameterRange<unsigned> mPPQRange;
};

}
```

```
#endif
#ifndef LADSPAMANAGER_H
#define LADSPAMANAGER_H

#include "AudioBuffer.hpp"
#include <iostream>
#include <sstream>
#if defined(__GNUC__) // GNU compiler
#include <dlfcn.h>
#endif
#include <ladspa.h>
#include <vector>

namespace tsal {

typedef std::pair<std::string, std::string> ladspa_key;

class LadspaManager {
  public:
    struct Port {
        Port(unsigned portId) : id(portId) {}
        AudioBuffer<LADSPA_Data> buffer;
        unsigned id;
    };
    static std::vector<ladspa_key> listPlugins();
    static const LADSPA_Descriptor* loadPlugin(const std::string& pluginPath);
};

}

#endif
#ifndef MIXER_H
#define MIXER_H

#include "OutputDevice.hpp"
#include "Channel.hpp"
#include "Context.hpp"
#include "Compressor.hpp"
#include "Sequencer.hpp"
#include "portaudio.h"
#include <memory>
#include <condition_variable>
#include <mutex>
```

```
namespace tsal {

/** @class Mixer
 * @brief The main audio manager that handles the overhead of audio buffers
 * and channel mixing
 *
 * To use TSAL, the Mixer class needs to be initiliazed at the start of the project.
 * All other audio devices will be routed through the Mixer's master channel
 */
class Mixer {
  public:
    Mixer();
    Mixer(unsigned sampleRate);
    ~Mixer();

    void add(Channel& channel);
    void add(Instrument& instrument);
    void add(Effect& effect);

    void remove(Channel& channel);
    void remove(Instrument& instrument);
    void remove(Effect& effect);

    Context getContext() { return mContext; };

    void requestModelChange(std::function<void()> change);
    void runModelChanges();

  private:
    bool mProcessing = false;
    unsigned mChangeRequests = 0;

    std::mutex mChangeRequestMutex;
    std::mutex mModelMutex;
    std::mutex mWaitModelChangeMutex;

    std::condition_variable mWaitModelChangeCondition;
    std::condition_variable mModelChangeRequestCondition;

    Context mContext;
    Channel mMaster;
    Compressor mCompressor;
```

```
    Sequencer mSequencer;
    PaStream *mPaStream;
    void openPaStream();

    AudioBuffer<float> mBuffer;
    int audioCallback(float *outputBuffer, unsigned long frameCount);
    static void paStreamFinished(void* userData);
    static int paCallback( const void *inputBuffer, void *outputBuffer,
                           unsigned long framesPerBuffer,
                           const PaStreamCallbackTimeInfo* timeInfo,
                           PaStreamCallbackFlags statusFlags,
                           void *userData );

};

}
#endif
#ifndef LADSPAEFFECT_H
#define LADSPAEFFECT_H

#include "Effect.hpp"
#include "AudioBuffer.hpp"
#include "LadspaManager.hpp"
#include <ladspa.h>

namespace tsal {

class LadspaEffect : public Effect {
  public:
    virtual void getOutput(AudioBuffer<float>& buffer) override;
    void loadPlugin(const std::string& pluginPath);
  private:
    LADSPA_Handle mHandle;
    const LADSPA_Descriptor * mDescriptor;
    std::vector<LadspaManager::Port> mInputPorts;
    std::vector<LadspaManager::Port> mOutputPorts;
    std::vector<LADSPA_Data> mControlPorts;
};

}

#endif
#ifndef SYNTH_H
```

```
#define SYNTH_H

#include "Oscillator.hpp"
#include "Envelope.hpp"
#include "Instrument.hpp"
#include "MidiNotes.hpp"
#include "Filter.hpp"
#include "Sequencer.hpp"

namespace tsal {

/** @class Synth
 * @brief A monophnoic synthesizer
 *
 * Synth uses the oscillator and envelope classes to create a basic
 * monophonic synthesizer that can be used to play midinotes
 */
class Synth : public Instrument {
  public:
    Synth();
    virtual void getOutput(AudioBuffer<float> &buffer) override;
    virtual void play(double note, double velocity = 100.0) override;
    virtual void stop(double note = MidiNote::A0) override;
    void setVelocity(double velocity);
    void setMode(Oscillator::OscillatorMode mode) { mOscillator.setMode(mode); };
    void setEnvelopeActive(bool active = true) { mEnvelope.setActive(active); };
    void setEnvelope(double attackTime, double decayTime, double sustainLevel,
                     double releaseTime) {
      mEnvelope.setActive();
      mEnvelope.setEnvelope(attackTime, decayTime, sustainLevel, releaseTime);
    };

    double getNote() const;

    virtual void updateContext(const Context& context) override {
      OutputDevice::updateContext(context);
      mOscillator.updateContext(context);
      mEnvelope.updateContext(context);
      mLFO.updateContext(context);
    }

  private:
    Oscillator mLFO;
```

```
    Filter mFilter;
    Oscillator mOscillator;
    Envelope mEnvelope;

    double mLfoAmount = 1;
    double mVelocity = 0.0;
    double mNote = 0.0;

    static Util::ParameterRange<double> mVelocityRange;
};

}

#endif
#ifndef OSCILLATOR_H
#define OSCILLATOR_H

#include "OutputDevice.hpp"
#include "ParameterManager.hpp"

namespace tsal {

/** @class Oscillator
 * @brief A device that generates sound from a waveform
 *
 * Oscillator is a low level synthesizer that generates samples from
 * algorithmic waveforms. It supports saw, sine, and square waveforms
 */
class Oscillator : public OutputDevice, public ParameterManager {
  public:
    Oscillator() :
      ParameterManager(OscillatorParameters) {};
    Oscillator(std::vector<Parameter> parameters) :
      ParameterManager(OscillatorParameters) {
      addParameters(parameters);
    };
    static std::vector<Parameter> OscillatorParameters;
    enum Parameters {
                OSCILLATOR_MODE = 0,
                MODULATION_MODE,
                MODULATION,
                FREQUENCY,
                PHASE_OFFSET,
```

```
  };
  enum ModulationMode {
                  NONE = 0,
                  MIX,
                  AM, // Amplitude modulation
                  PM, // Phaser modulation
  };
  /**
   * @brief Modes for the oscillator
   *
   * Oscillator mode can be switched at anytime during execution
   */
  enum OscillatorMode {
    SINE = 0,
    SAW,
    SQUARE,
    WHITE_NOISE,
  };

  double getSample();
  void setMode(OscillatorMode mode);
  void setNote(double note);
  void setFrequency(double frequency);
  virtual void updateContext(const Context &context) override {
    OutputDevice::updateContext(context);
    // Frequency parameter depends on sample rate
    parameterUpdate(FREQUENCY);
  }

  double getFrequency();
  unsigned getNote();

  static unsigned getNoteFromFrequency(double frequency);
  static double getFrequencyFromNote(double note);

protected:
  virtual void parameterUpdate(unsigned id) override;
private:
  double polyBLEP(double t);

  double mPhase = 0.0;
  double mPhaseStep = 0.0;
```

};

}
#endif
 # TSAL

A Thread Safe Audio Library that uses audiolizations to assist in the understanding of concurrent and parallel processing.

Getting Started

TSAL uses the GNU Autotools build system to build and install the library.

Prerequisites

Install [PortAudio](http://www.portaudio.com/)

Install autotools (`autoconf`, `automake`, and `libtool`) or cmake. For Windows, a build environment like MSYS2 is recommended

Build

Once the prerequisites have been met, building TSAL should follow the Autotools workflow

```console
autoreconf -vi
./configure
make
```

If you are using cmake,

``` console
mkdir build
cd build
cmake ..
make
```

This will compile all the necessary files, tests, and examples for the project
To install the library on your system run `make install` as root.

Running the tests

Use `make check` to run the tests

Built With

* [PortAudio](http://www.portaudio.com/) - Audio API
* [Midifile](https://github.com/craigsapp/midifile) - MIDI processing
* [TinySoundFont](https://github.com/schellingb/TinySoundFont) - SoundFont player

License